

Digital Signal Processing Applications with

MOTOROLA'S DSP56002 Processor



Mohamed El-Sharkawy, Ph.D.

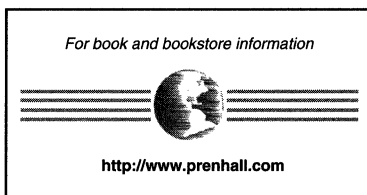


Digital Signal Processing Applications with Motorola's DSP56002 Processor

Mohamed El-Sharkawy, Ph.D.

Purdue University

*With Appendices provided by the Applications
Engineering Staff of Motorola's DSP Operation*



Prentice Hall PTR
Upper Saddle River, NJ 07458

Library of Congress Cataloging-in-Publication Data

El-Sharkawy, Mohamed.

Digital signal processing applications with Motorola's DSP56002 processor / by Mohamed el-Sharkawy; with appendices provided by the applications engineering staff of Motorola's DSP Operation.

p. cm.

Includes bibliographical references.

ISBN 0-13-569476-0 (hardcover)

1. Signal processing—Digital techniques. 2. Motorola DSP56002 (Microprocessor). I. Motorola, inc. II. Title.

TK5102.9.E37 1996

621.382'2'02854165—dc20

96-13223

CIP

Editorial/production supervision: *BooksCraft, Inc., Indianapolis, IN*

Cover design director: *Jerry Votta*

Cover design: *Paul Gourhan*

Acquisitions editor: *Karen Gettmann*

Manufacturing manager: *Alexis R. Heydt*



© 1996 by Prentice Hall PTR

Prentice-Hall, Inc.

A Simon & Schuster Company

Upper Saddle River, NJ 07458

The publisher offers discounts on this book when ordered in bulk quantities.

For more information, contact:

Corporate Sales Department

Prentice Hall PTR

One Lake Street

Upper Saddle River, NJ 07458

Phone: 800-382-3419 Fax: 201-236-7141

E-mail: corpsales@prenhall.com.

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

All product names mentioned herein are the trademarks of their respective owners.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN: 0-13-569476-0

Prentice-Hall International (UK) Limited, *London*

Prentice-Hall of Australia Pty. Limited, *Sydney*

Prentice-Hall Canada Inc., *Toronto*

Prentice-Hall Hispanoamericana, S.A., *Mexico*

Prentice-Hall of India Private Limited, *New Delhi*

Prentice-Hall of Japan, Inc., *Tokyo*

Simon & Schuster Asia Pte. Ltd., *Singapore*

Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Table of Contents

Preface	xiii
1 Introduction to the DSP56002 Digital Signal Processor and the DSP56002EVM Evaluation Module	1
1.1 DSP56002 Processor General Description	1
1.1.1 Digital Signal Processing Core	3
1.1.2 Expansion Area	3
1.2 DSP56002EVM Evaluation Module General Description	4
1.3 EVM Set Up	4
1.4 Installing the Software	5
1.5 Summary of Debug-EVM Commands	5
1.6 Basics of Some Debug-EVM Commands	6
1.6.1 Help Command	6
1.6.2 Radix Command	9
1.6.3 Display Command	10
1.6.4 Change Command	11
1.6.5 Copy Command	15
1.7 Using the EVM to Program the DSP56002	15
1.7.1 Enter, Assemble, and Disassemble a Canned DSP56002 Program	15
1.7.2 Saving and Loading or Reloading a Canned DSP56002 Program	17
1.7.3 Testing and Executing a Canned DSP56002 Program	18
1.8 Reference	24
2 Introduction to Motorola's DSP Assembler Program	25
2.1 Installing the ASM56000 Assembler Software Program	25
2.2 Program Source Statement Format	26
2.3 Entering and Editing a Canned DSP56002 Program	28
2.4 Assembling a Canned DSP56002 Program	29
2.5 Program Development and Execution Using the Debug-EVM Software Program	30

2.5.1 More Debug-EVM Software Program Basics	30
2.5.2 More on Using the Debug-EVM Software Program to Load, Execute, and Test a Canned DSP56002 Program.....	33
2.6 Defining and Using Macros	37
2.6.1 Programming a Canned DSP56002 Macro.....	38
2.6.2 Assembling a Canned DSP56002 Macro	40
2.6.3 Loading a Canned DSP56002 Macro	41
2.7 References.....	41
3 DSP56002 Architecture and Addressing Modes	43
3.1 DSP56002 Architecture Review	43
3.1.1 DSP56K Central Processing Unit.....	44
3.1.2 Expansion Area.....	44
3.2 How to Proceed.....	45
3.3 Data ALU Execution Unit	45
3.3.1 Data Registers	46
3.3.2 MAC and Logic Unit.....	50
3.3.3 Accumulator Shifter	55
3.3.4 Data Shifter/Limiters	56
3.3.5 Bit Manipulation Unit.....	61
3.4 Program Controller	62
3.5 Address Generation Unit.....	66
3.6 Addressing Modes	70
3.6.1 Register Direct Addressing Modes.....	70
3.6.2 Special Addressing Modes.....	71
3.6.3 Address Register Indirect Addressing Modes	76
3.6.4 DSP56002 Addressing Mode Summary.....	82
3.7 Reference	82
4 DSP56002 Instruction Set.....	85
4.1 Instruction Format.....	85
4.1.1 One-Word Instruction.....	85
4.1.2 Two-Word Instruction	86
4.2 Parallel-Move Operations.....	87
4.3 Parallel-Move Types.....	87
4.3.1 Immediate Short Data Move	87
4.3.2 Register to Register Data Move	89
4.3.3 Address Register Update.....	90
4.3.4 X- or Y-Memory Data Move.....	91
4.3.5 X- or Y-Memory and Register Data Move	92
4.3.6 Long Memory Data Move	93
4.3.7 XY-Memory Data Move	94
4.3.8 Parallel-Moves Summary.....	95
4.4 DSP56002 Instruction Set.....	96
4.4.1 Move Instructions	97

4.4.2 Arithmetic Instructions.....	99
4.4.3 Logic Instructions.....	110
4.4.4 Bit Manipulation Instructions.....	113
4.4.5 Loop Instructions.....	114
4.4.6 Program Control Instructions.....	118
4.4.7 DSP56002 Instruction Set Summary.....	121
4.5 Reference.....	124
5 Introduction to Digital Signal Processing Systems.....	125
5.1 DSP System.....	126
5.2 CS4215 Multimedia Audio Codec	127
5.2.1 Control Mode	128
5.2.2 Data Mode.....	133
5.3 DSP56002 Processor Port C	135
5.3.1 General-Purpose I/O (GPIO).....	135
5.3.2 DSP56002 Processor SSI Port Pins	139
5.3.3 SSI Port Selection.....	139
5.3.4 SSI Port Control Register “A” (CRA).....	140
5.3.5 SSI Port Control Register “B” (CRB).....	143
5.3.6 SSI Port Receive Registers	144
5.3.7 SSI Port Transmit Registers.....	145
5.3.8 SSI Port Status Register	145
5.4 DSP56002 Interrupt Priority Register	147
5.5 Initialization Macro	151
5.6 Exercising the DSP System.....	155
6 Designing FIR Filters and Implementing Them	
on the DSP56002 Processor	157
6.1 FIR Filter Frequency Response	158
6.2 Relating a Practical FIR Filter to an Ideal FIR Filter.....	158
6.3 Specifying a Practical FIR Filter	161
6.4 Using the Window Method to Design a Practical FIR Filter.....	162
6.5 Using a Software Program to Design Practical FIR Filters.....	165
6.5.1 Low-Pass Filter Design Example	166
6.5.2 High-Pass Filter Design Example	167
6.5.3 Band-Pass Filter Design Example	168
6.5.4 Band-Stop Filter Design Example.....	169
6.6 Implementing a FIR Filter.....	170
6.6.1 Implementing a FIR Filter on the DSP56002 Processor.....	170
6.6.2 DSP56002 Instructions for Implementing a FIR Filter	171
6.6.3 DSP56002 Macros for Implementing a FIR Filter	
on the DSP System	177
6.7 Digital Filter Design and Analysis System Software Program.....	179
6.8 FIR Filter Demonstration System	179

6.9 Implementing FIR Filters	180
6.9.1 Implementing Low-Pass FIR Filter	180
6.9.2 Implementing a High-Pass FIR Filter	188
6.9.3 Implementing a Band-Pass FIR Filter	199
6.9.4 Implementing a Band-Stop FIR Filter	211
6.10 References	219
7 Designing IIR Filters and Implementing Them on the DSP56002 Processor	221
7.1 IIR Filter Transfer Function	222
7.2 Review of the Bilinear Transformation	224
7.3 IIR Filter Specifications	227
7.4 Normalized Low-Pass Filter Design	228
7.4.1 Butterworth Filter Design	232
7.4.2 Chebyshev Filter Design	234
7.5 Analog Filter Design	236
7.6 Digital Filter Design	237
7.7 Implementing a Biquad Section on the DSP56002 Processor	237
7.8 Implementing an IIR Filter	244
7.9 Digital Filter Design and Analysis System Package	245
7.10 DSP56002 Macro for Implementing an IIR Filter on the DSP System	245
7.11 IIR Filter Demonstration System	246
7.12 Designing and Implementing IIR Filters	246
7.12.1 Designing and Implementing a Low-Pass IIR Filter	247
7.12.2 Designing and Implementing a High-Pass IIR Filter	255
7.12.3 Designing and Implementing a Band-Pass IIR Filter	262
7.12.4 Designing and Implementing a Band-Stop IIR Filter	271
7.13 Quantization Effects	279
7.14 References	282
8 Implementing the Fast Fourier Transform with the DSP56002 Processor	283
8.1 FFT Review	284
8.1.1 Butterfly Computation	288
8.1.2 "Bit-Reversed" Reordering of Input Data	289
8.2 Running the DSP56002 Demonstration Software	289
8.3 FFT Implementation	289
8.3.1 Generating Twiddle Factors	290
8.3.2 Storing the Input Sequence $X(n)$ in Bit-Reversed Order	292
8.3.3 Butterfly Computation	296
8.3.4 Group and Coefficient Offsets	305
8.3.5 FFT Macros	306
8.4 References	317

9 Designing Adaptive FIR Filters and Implementing Them on the DSP56002 Processor	319
9.1 LMS Algorithm Review	320
9.2 Implementing an Adaptive FIR Filter on the DSP56002	323
9.2.1 Implementing an Adaptive FIR Filter on the DSP56002 Processor	323
9.2.2 DSP56002 Instructions for Implementing an Adaptive FIR Filter	324
9.3 Adaptive FIR Filter Demonstration System	334
9.4 Implementing an Adaptive FIR Filter on the Demonstration System	334
9.5 References	342
A DSP56001 Interface Techniques and Examples	351
A.1 Interfacing Motorola's DSP56001 to Pseudo Static RAM	351
A.2 Pseudo-Static RAM	351
A.2.1 DSP56001 Memory I/O Basics	352
A.2.2 Memory Subsystem Overview	353
A.2.3 Circuit Description	355
A.2.4 Summary	356
A.3 A Simple Dynamic RAM Interface for the DSP56001	360
A.4 DRAM Basics	360
A.4.1 Circuit Overview	362
A.4.2 Circuit Description	365
A.4.3 Summary	366
A.5 A Simple ISA Bus Interface for the DSP56001	369
A.5.1 Interface Circuit Overview	370
A.5.2 Detailed Circuit Description	372
A.5.3 Timing	373
A.6 Communicate with the DSP56000 Host Interface Using C Language	373
A.6.1 Introduction	373
A.6.2 Example Program	374
A.7 References	377
B Implementing IIR/FIR Filters with Motorola's DSP56000/DSP6001	379
B.1 Introduction	379
B.1.1 Analog RCL Filter Types	381
B.1.2 Analog Lowpass Filter	381
B.1.3 Analog Highpass Filter	384
B.1.4 Analog Bandstop Filter	385
B.1.5 Analog Bandpass Filter	387
B.2 Second-Order Direct-Form IIR Digital Filter Sections	390
B.2.1 Digital Lowpass Filter	394

B.2.2 Digital Highpass Filter	398
B.2.3 Digital Bandstop Filter	401
B.2.4 Digital Bandpass Filter	404
B.2.5 Summary of Digital Coefficients	406
B.3 Single-Section Canonic Form (Direct Form II)	409
B.3.1 The Canonic-Form Difference Equation	409
B.3.2 Analysis of Internal Node Gain	411
B.3.3 Implementation on the DSP56001	414
B.4 Single-Section Transpose Form	414
B.4.1 Gain Evaluation of Internal Nodes	414
B.4.2 Implementation on the DSP56001	420
B.5 Cascaded Direct Form	421
B.5.1 Butterworth Lowpass Filter	421
B.5.2 Cascaded Direct-Form Network	424
B.6 Filter Design and Analysis System	427
B.6.1 Canonic Implementation	429
B.6.2 Transpose Implementation (Direct Form I)	435
B.7 FIR Filters	439
B.7.1 Linear-Phase FIR Filter Structure	440
B.7.2 Linear-Phase FIR Filter Design Using the Frequency Sampling Method	444
B.7.3 FIR Filter Design Using FDAS	452
B.7.4 FIR Implementation on the DSP56001	456
B.8 References	458
C Implementation of Fast Fourier Transforms on Motorola's Digital Signal Processors	461
C.1 Introduction to the Fourier Integral	461
C.1.1 Definition and History	461
C.1.2 Use of the Fourier Transform	462
C.2 The Discrete Fourier Transform	464
C.2.1 The Discrete-Time Fourier Transform (DTFT)	464
C.2.2 Windowing and Windowing Effects	466
C.2.3 Sampling the Frequency Function	467
C.3 The Fast Fourier Transform	469
C.3.1 Motivation	469
C.3.2 Divide and Conquer	469
C.3.3 The Decimation-in-Time and Decimation-in-Frequency Radix-2 Fast Fourier Transforms	470
C.3.4 The Decimation-in-Frequency Radix-2 Fast Fourier Transforms	474
C.4 Complex FFT on the Motorola DSP Family	474
C.4.1 Required Hardware Support for FFT Calculation	474
C.4.2 Radix-2 DIT and DIF Butterflies	475
C.4.3 Complexity of a Radix-2 DIT FFT	476

C.4.4 Implementation on Motorola's DSP56001.....	477
C.4.5 Implementation on Motorola's DSP96002.....	481
C.4.6 Implementation on Motorola's DSP56156.....	483
C.4.7 Scaling for Fixed-Point Processors (DSP56001/2 and DSP56156).....	485
C.4.8 Twiddle Factors and On-Chip ROM	487
C.4.9 Bit-Reversed Addressing	488
C.4.10 Implementation of a Radix-4 DIT FFT on DSP96002	488
C.4.11 Inverse FFT.....	492
C.5 Optimizing Performance of the FFT	493
C.5.1 Optimization	493
C.5.2 Minimum Memory Requirement — In-Place Calculation.....	494
C.5.3 Optimization for Faster Execution	494
C.5.4 Example of Optimization.....	497
C.6 Real-Valued Input FFT Algorithm	499
C.6.1 Real-Valued Input FFT Algorithm 1	499
C.6.2 Real-Valued Input FFT Algorithm 2	504
C.6.3 Real-Valued Input FFT Algorithm 3	507
C.6.4 The Goertzel Algorithm.....	508
C.6.5 Real-Time Data Acquisition on Motorola DSPs.....	510
C.7 Two Dimensional Fourier and Cosine Transforms.....	512
C.7.1 Two Dimensional FFTs on the DSP96002.....	512
C.7.2 Discrete Cosine Transform on the DSP96002.....	512
C.8 Competitive Analysis of FFT Performances	514
C.8.1 Most Popular Digital Signal Processors	514
C.8.2 Performance of FFTs on Digital Signal Processors.....	514
C.9 Conclusion.....	517
C.9.1 Acknowledgments	518
C.10 Fully Optimized Complex FFT.....	518
C.10.1 Optimized Complex FFT for the DSP96002.....	518
C.11 Real-Valued Input FFT	534
C.11.1 Faster Real FFT for the DSP96002	534
C.11.2 Real FFT for DSP56001/2.....	536
C.12 References.....	540
Index.....	541
DSP56002 Demonstration Software Order Form	545

Preface

The objective of this book is to explain and demonstrate the operation of the DSP56002 Processor and its Development Tools, which are used to implement solutions to common real-time digital signal processing problems. This book is intended for use by both undergraduate and graduate engineering students, as well as industry professionals. It can be used, for example, in a one-semester, three-credit-hour digital signal processing projects course or in a laboratory course accompanying a first course in digital signal processing.

This book consists of three major parts. The first, including chapters 1 and 2, explains and demonstrates the operation of DSP56002 digital signal processor, the DSP56000EVM Evaluation Module (EVM), and the DSP Assembler Software program. The second part, including chapters 3 and 4, describes the DSP56002 processor's architecture, addressing modes, and instruction set and shows how they operate. The third part, which includes chapters 5, 6, 7, 8, and 9, explains, implements, and demonstrates common real-time digital signal processing routines.

This book is the result of the generous support of Motorola's Digital Signal Processor Division and its DSP University Support Program. In particular, I thank Jim George, Keith Essency, and Robert M. Bergeler of Motorola's DSP for supporting the DSP laboratory at Purdue University. In addition, I thank Robert M. Bergeler for his significant efforts in carefully editing and revising the first draft of the book, helping to clarify my ideas. I also gratefully acknowledge my students for their commentary and assistance throughout my writing of the book. Finally, I thank Jerry Purcell, president and technical director of Momentum Data Systems Incorporated, for permission to describe the use of the Digital Filter Design and Analysis System program in the book.

The minimum hardware configuration to run the DSP56002 Development Tools and the DSP56002 Demonstration Software program is an IBM PC with:

- A. 386 or 486 processor
- B. 512 Kbytes of RAM
- C. Hard drive

- D.** One RS-232 Port
- E.** 3.5" 1.44MB diskette drive
- F.** MS-DOS V3.0 or later

A highly recommended software program is available for use with this book. The *DSP56002 Demonstration Software* program includes all of the DSP56002 programs presented in the book. The program is sold for \$25. Please complete the enclosed order form in the back of the book and mail it along with a check or money order to the address on the form.

Introduction to the DSP56002 Digital Signal Processor and the DSP56002EVM Evaluation Module

The DSP56002 is a general purpose, single chip Digital Signal Processor (DSP). The DSP56002EVM Evaluation Module (EVM) is a low-cost platform that uses the DSP56002 processor for designing real-time DSP systems.

In this chapter, a general description of the DSP56002 processor is presented, followed by a general description of the DSP56002EVM Evaluation Module. Step-by-step instructions covering the software installation of the Domain Technologies' Windowed Interface Debug EVM Software (Debug-EVM) are provided. Debug-EVM Commands are then described and used to assemble, execute, and test a DSP56002 object code program.

1.1 DSP56002 PROCESSOR GENERAL DESCRIPTION

The DSP56000 family of processors is composed of an efficient 24-bit digital signal processor core and an expansion area. In the expansion area, the chip can support various configurations of memory and peripheral modules that may change from family member to family member.

The DSP56002 is a member of the DSP56000 family of processors where the 56000-family-compatible digital signal processing core is fed by on-chip program RAM, two independent data RAMs, and two data ROMs with sine and μ -law and A-

law tables. The DSP56002 contains a Serial Communication Interface (SCI), Synchronous Serial Interface (SSI), parallel Host Interface (HI), Timer/Event Counter, Phase-Locked Loop (PLL), and On-chip Emulation (OnCE) port. Figure 1.1 shows a block diagram of the DSP56002, including the DSP core and the expansion area for memory and peripherals.

The DSP56002 processor offers abundant features that combine to provide the processing strength necessary for efficient realization of high performance DSP or microcontrol systems.

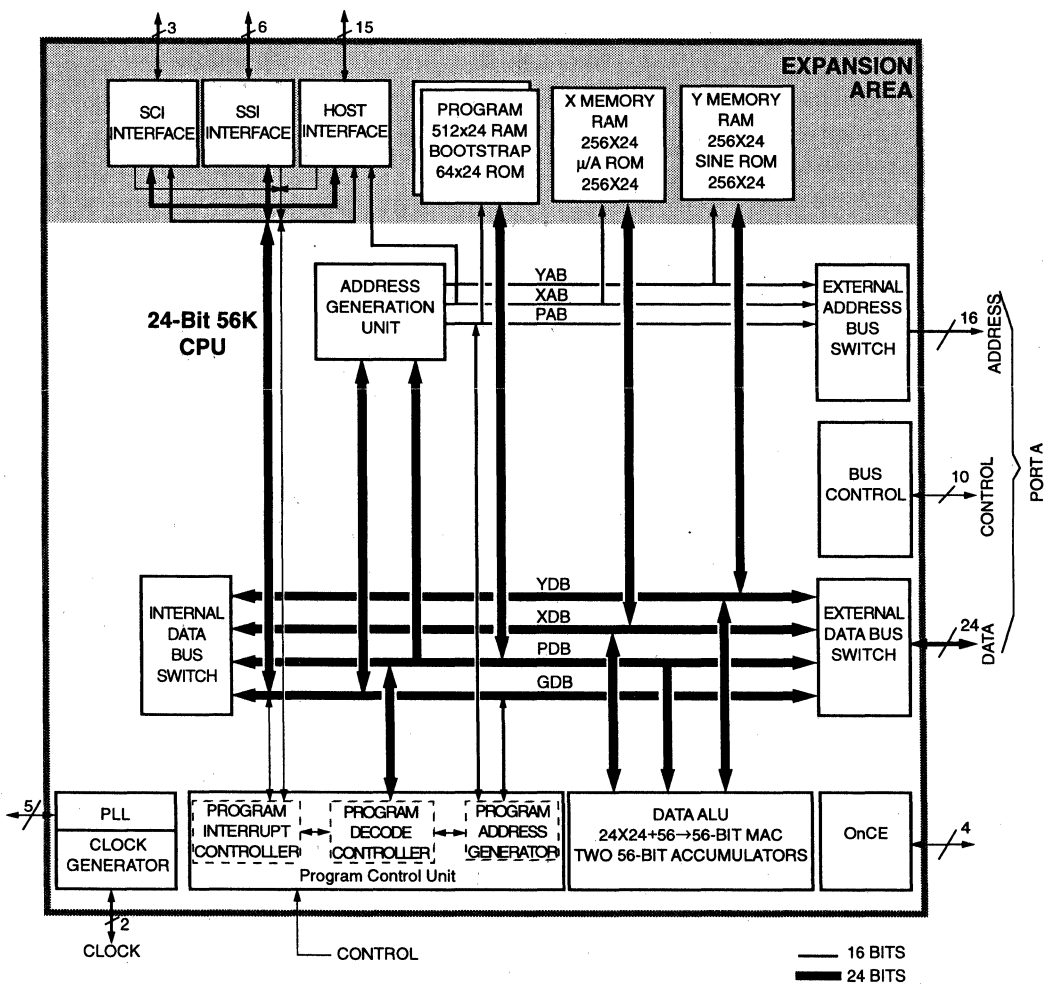


Fig. 1.1 DSP56002 Block Diagram

1.1.1 Digital Signal Processing Core

A. The major components of the digital signal processing core are:

1. *Three independent Execution Units:*

- a.** the Data Arithmetic Logic Unit (data ALU)
- b.** the Address Generation Unit (AGU)
- c.** the Program Control Unit (PCU)

2. *Four independent 24-bit Data Buses:*

- a.** the X Data Bus (XDB)
- b.** the Y Data Bus (YDB)
- c.** the Program Data Bus (PDB)
- d.** the Global Data Bus (GDB)

3. *Three independent 16-bit Address Buses:*

- a.** the X Address Bus (XAB)
- b.** the Y Address Bus (YAB)
- c.** the Program Address Bus (PAB)

4. Memory Expansion Port (Port A)

5. On-Chip Emulator (OnCE) circuitry

6. Phase-locked Loop (PLL) based clock circuitry

B. The DSP core main features include:

- 1.** 40 Million Instructions Per Second (MIPS) at 80 MHz
- 2.** 240 Million Operations Per Second (MOPS) at 80 MHz
- 3.** Highly parallel instruction set with unique DSP addressing modes
- 4.** Parallel 24×24-bit multiply-accumulate in one instruction cycle (two clock cycles)
- 5.** Zero overhead nested DO loops
- 6.** Fast auto-return interrupts
- 7.** Very low-power CMOS design
- 8.** STOP and WAIT low-power standby modes

1.1.2 Expansion Area

The major components in the expansion area are:

- 1.** 512×24 Program RAM
- 2.** Two 256×24 Data RAM
- 3.** Two 256×24 Data ROM
- 4.** Byte-wide Host Interface with DMA Support
- 5.** Synchronous Serial Interface Port
- 6.** Serial Communication Interface (Asynchronous) Port

1.2 DSP56002EVM EVALUATION MODULE GENERAL DESCRIPTION

The DSP56002EVM Evaluation Module (EVM) is a hardware tool for designing, debugging, and evaluating DSP56002-based systems. As shown in Fig. 1.2, the EVM consists of three components:

1. A DSP56002 Evaluation Board that contains a DSP56002 processor, off-chip expansion memory, stereo analog-to-digital and digital-to-analog converter, interface and control circuitry, and several connectors for external access
2. Motorola's DSP5600x Assembler
3. Domain Technologies' Windowed Interface Debug EVM Software (Debug-EVM)

The software runs under MS-DOS on an IBM PC and communicates with the EVM over an RS-232 serial port. The user must provide a power supply (7–9 V AC or DC, 700 mA) and RS-232 cable with DB-9 connectors.

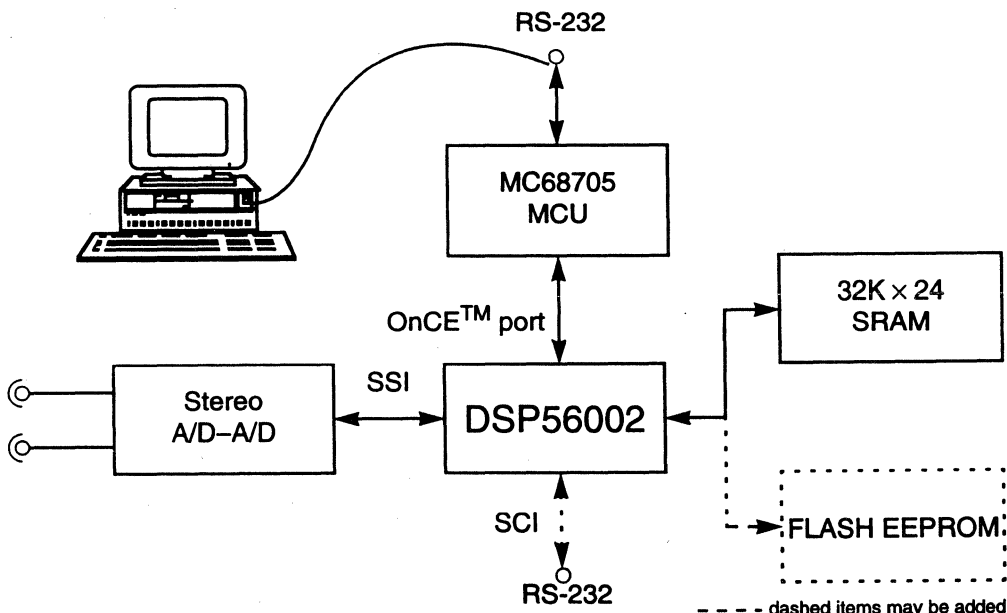


Fig. 1.2 DSP56002 EVM Block Diagram

1.3 EVM SET UP

1. Turn off the power to your PC and unplug it from its power source.
2. Connect the RS-232 cable between the PC's serial port and the EVM's RS-232 connector.
3. Connect a 7–9 volt calculator-style power supply to the EVM's 2.1 MM jack (or the two screw terminals).

4. Plus your PC and the EVM's power supply into their respective power sources.
5. Turn on the power to your PC.

1.4 INSTALLING THE SOFTWARE

The DSP56002EVM comes with Motorola's DSP5600X Assembler and Domain Technologies' Windowed Interface EVM Software (Debug-EVM), offering symbolic addressing and animated ASCII graphical display of memory segments.

For simplicity it is assumed that Drive "C" is assigned to the hard disk and that DOS resides in directory c:\dos.

1. Boot the PC.
2. Place the diskette containing the software into Drive "A."
3. Type `md c:\evm<return>` to create an `evm` directory on Drive "C."

Optionally, you can give the `evm` directory any name you wish and locate it anywhere you wish on Drive "C."

4. Type `copy a: c:\evm<return>` to copy the contents of the diskette in "A" to the specified directory on Drive "C."
5. Type `cd c:\evm<return>` to enter this directory.
6. Type `evm56k<return>` to enter the Debug-EVM program.

The following four windows should appear at your screen (see Fig. 1.3):

1. the Command
2. the Data1
3. the Unassemble
4. the Registers Windows

Note also that the command window has double line borders to indicate that the command window is the window selected. The cursor is inside the command window.

7. Type `quit<return>` to exit the Debug-EVM program.

1.5 SUMMARY OF DEBUG-EVM COMMANDS

The Debug-EVM commands are permuted by several optional parameters specific to each command to yield a large and powerful command set. This book does not attempt to use all of the numerous combinations of commands and specific optional parameters. Rather, only those commands and command parameters necessary to demonstrate the concepts of this book are covered. For detailed information on the Debug-EVM commands and their optional parameters, refer to Domain Technology's *User's Manual*. The Debug-EVM Commands (without optional parameters) are summarized in Table 1.1.

```

[Data]HEX1( ) [11]
X:012C A0FB10 007700 79DD5B FF7DFF 00B000 000200
X:0132 60FF00 601F21 829F20 B53C00 AF7700 BFAF88
X:0138 000000 005700 65FB33 FFFF27 3C3700 804F88
X:013E DEFFD6 FFFFA0 7FFEDF F636FF 7700EF BA50FD
X:0144 FFFFFF FF40FF BF1CFB BF00FF FF50FF D9C8FF
X:014A 0D0067 44001F FF6BFD F79AFF 5E60FF 1600FD

[Registers]HEX1 [11]
R:FFFFFFFF
Y:FFFFFFFF000000
X1:FFFFFF X0:FFFFFF
Y1:FFFFFF Y0:000000
A:FFFFFFFFFFFFFF
B:00000000000000
02:FF 01:FFFFFF 00:FFFFFF
02:00 01:000000 00:000000
00:01FF 00:FFFF 00:FFFF
01:0000 01:FFFF 01:FFFF
02:FFFF 02:FFFF 02:FFFF
03:E963 03:FFFF 03:FFFF
04:FFFF 04:FFFF 04:FFFF
05:FFFF 05:FFFF 05:FFFF
06:FFFF 06:FFFF 06:FFFF
07:E963 07:FFFF 07:FFFF
08:1400 0C:A32F 0P:00
0R:0314 0C:0201 0NH:00

[Unassemble] [11]
P:0201 00C501 DC $00C501
P:0202 40DF40 ADD X0,A L:(R7)+,A10
P:0203 357F45 CMP X0,A #57F,A5
P:0204 E7EF01 TFR B,A X:(R7)+M7,X1 Y:
P:0205 56FF40 ADD X0,A X:-(R7),A
P:0206 AFFF40 ADD X0,A X:(R7)+,B A,Y:
P:0207 AFFFF4 MPY -Y1,X1,A X:(R7)+,B
P:0208 915A01 TFR B,A X0,X:(R2)+ Y1,Y
P:0209 005E00 DC $005E00
P:020A 71FD00 MOVE X:-(R5),M1

[Command]HEX1 [11]
EVM>

Menu A:PRG61.LOD [Step:OFF Upd:OFF Step Imp PG:0201]

```

Fig. 1.3 Screen Display After Entering the Debug-EVA Program

1.6 BASICS OF SOME DEBUG-EVM COMMANDS

1.6.1 Help Command

The **help** command allows the user to review the Debug-EVM Commands or get information on a particular command.

1. Type `help<return>` to display a listing of the Debug-EVM commands.

The display on your screen should be like that in Fig. 1.4.

2. Use the “up arrow” and “down arrow” keys to survey the Debug-EVM commands.
3. Hit `<escape>`.
4. Type `help display<return>` to display a listing of the display command options.

The display on your screen should be like that in Fig. 1.5.

5. Hit `<escape>`.
6. Type `help assemble<return>` to display a listing of the assemble command options.

The display on your screen should be as shown in Fig. 1.6.

7. Use the `help` command to survey the `go`, `break`, and `trace` commands, e.g., `help go<return>`, `help break<return>`, `help trace<return>`.

Table 1.1 Summary of Debug-EVM Commands

Command	Description
ALIAS	Define custom command string
ASSEMBLE	On screen assembler
BREAK	Set breakpoint
CHANGE	Modify registers/memory values
CONFIG	System configuration options
COPY	Copy memory block to another place
DISPLAY	Display memory values
DISASSEMBLE	Disassemble memory
FORCE	Reset or stop DSP
GO	Execute DSP program
HELP	Display help screen
JUMP	Jump over a subroutine call
LOAD	Load DSP program
LOG	Log all commands to the log file
PATH	Define or display file directory path
QUIT	Quit Debug-EVM program
RADIX	Change default input number base
SAVE	Save memory blocks to file or state
STEP	Single step DSP program
SYMBOL	Display symbol table
SYSTEM	Execute operating system command(s)
TRACE	Trace through DSP program
UNASSEMBLE	Unassemble memory
UNALIAS	Remove custom command string
USE	Use different directory
VERSION	Display revision of EVM's firmware
VIEW	Open text file for viewing
WAIT	Wait before continuing
WATCH	Select variable to watch
WINDOW	Windows control options

```

[Data] [HEX] ( ) [F1] [Registers] [HEX] [F1]
X:012C A0FB10 007700 79DD5B FF7DFF 00B000 00B200 X:FFFFFFFFFFFF
EVM-56K Help 1
Function key definition:      Alt key definition:
F1: help                    Alt-X - exit to DOS
F2: change data format      Alt-N - main menu
F3: change data mode        Alt-1..4 - Data windows
F4: move/size window        Alt-C - Command window
F5: go to cursor            Alt-F - Flag window
F6: toggle data refresh     Alt-I - I/O window
F7: trace                   Alt-O - OnCE window
F8: jump                    Alt-R - Registers window
F9: toggle breakpoint       Alt-S - Stack window
F10: start/stop execution   Alt-T - Text window
                             Alt-U - Unassemble window
                             Alt-W - Watch window
SF7: toggle continuous trace
SF8: toggle continuous jump

Available Commands:
ALIAS [name] [key] ["command string"]
- assign command string to a name or function key
Supported keys: F1..F12, SF1..SF12, AF1..AF12, CF1..CF12

HELP

```

Fig. 1.4 Partial Listing of the Debug-EVM Commands

```

[Data] [HEX] ( ) [F1] [Registers] [HEX] [F1]
X:012C A0FB10 007700 79DD5B FF7DFF 00B000 00B200 X:FFFFFFFFFFFF
EVM-56K Help 1
DISPLAY [address] [-hi-di-fi-b] [-a [-graphi-text]]
- display P/X/Y/L memory in data window using selected radix

HELP [command]

```

Fig. 1.5 Listing of the Display Command Options

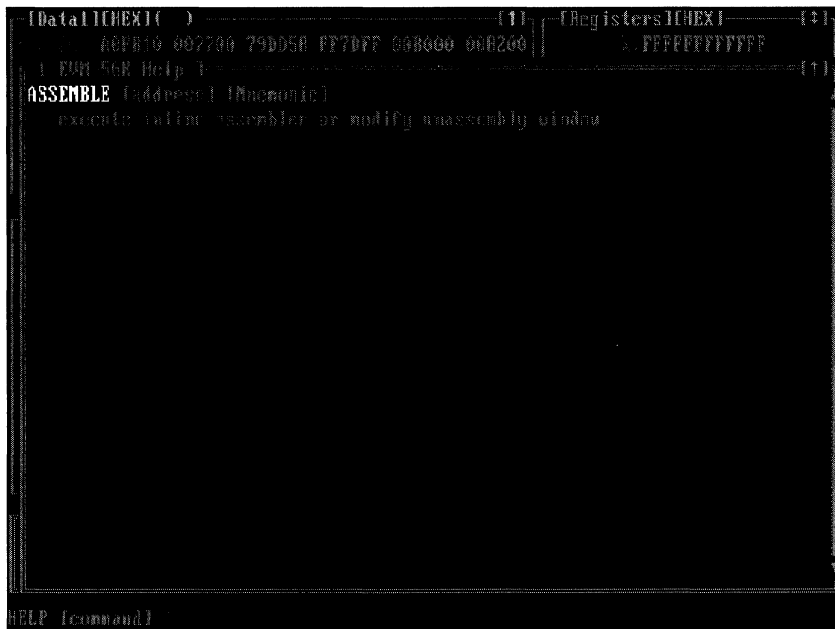


Fig. 1.6 Listing of the Assemble Command Options

1.6.2 Radix Command

The **radix** command changes the default radix of the command window. All constants (data and addresses) entered in the command window are accepted according to the set radix. The radix may be overwritten if one of the following symbols precedes a number: The dollar symbol (\$) for hexadecimal numbers, the grave accent symbol (`) for decimal numbers, and the percent symbol (%) for binary numbers.

For example, if you need to enter a large number of hexadecimal constants, the **radix** command can be used to change the default radix to hexadecimal. However, after doing so you need to precede decimal constants with a grave accent (`) radix specification symbol and precede binary constants with a percent (%) radix specification symbol.

1. Type `radix b<return>` to change the default radix to binary.

Note that the radix is displayed in the upper-left corner of the command window.

2. Hit <f2> to change the default radix to decimal.
3. Hit <f2> to change the default radix to hexadecimal.
4. Hit <f2> to change the default radix to fractional.
5. Hit <f2> to change the default radix to binary.
6. Type `radix h<return>` to change the default radix to hexadecimal.

1.6.3 Display Command

The **display** command can be used to display program or data memory in the data window.

1. Type `display p:$100<return>` to display the values of P-Memory starting from location 100 hexadecimal.

The display on your screen should be similar to the one in Fig. 1.7.

Note that the program memory values are displayed in the data window. Note also that the DSP's internal registers are already displayed in the registers window. The notations used for the registers currently displayed in the registers window are:

- a. DSP56002 Data ALU Registers:
x,x1,x0,y,y1,y0 for input registers
a,a2,a1,a0,b,b2,b1,b0 for accumulator registers
- b. DSP56002 Address Generation Unit Registers:
r0-r7 for address registers
n0-n7 for offset registers
m0-m7 for modifier registers
- c. DSP56002 Program Controller Registers:
pc for Program Counter register
sr for Status Register

[Data] [HEX] () [11]			[Registers] [HEX] [11]		
P:0100	FFFFFF	FFFFFF	X:FFFFFF	FFFFFF	
P:0106	FFFFFF	FFFFFF	Y:FFFFFF	000000	
P:010C	FFFFFF	FFFFFF	X1:FFFFFF	X0:FFFFFF	
P:0112	FFFFFF	FFFFFF	Y1:FFFFFF	Y0:000000	
P:0118	FFFFFF	FFFFFF			
P:011E	FFFFFF	FFFFFF			
[Unassemble] [11]					
P:0201	00C501	DC \$00C501			
P:0202	40DF40	ADD X0,A L:(R7)+,A10			
P:0203	357F45	CMF X0,A #57F,R5			
P:0204	E7EF01	TFH B,A X:(R7)+N7,X1 Y:			
P:0205	56FF40	ADD X0,A X:-(R7),A			
P:0206	A0FF40	ADD X0,A X:(R7)+,B A,Y:			
P:0207	A0FFP4	MPY -Y1,X1,A X:(R7)+,B			
P:0208	915A01	TFH B,A X0,X:(R2)+ Y1,Y			
P:0209	005E00	DC \$005E00			
P:020A	71FD00	MOUE X:-(R5),N1			
[Command] [HEX] [11]					
EVM: DISPLAY p:\$100					
EVM:					
[1] A:PROG1.LOD			[1] LA:1400 LC:A32F SP:00		
			SR:0314 PC:0201 DMR:00		
			[1] CS:OFF Upd:OFF Stp Jmp PC:0201		

Fig. 1.7 Values of P-Memory Starting from Location 100 Hexadecimal After the Execution of the Display Command

omr for Operating Mode Register

la for Loop Address register

lc for Loop Counter register

sp for Stack Pointer register

2. Type `display y:$100 -b -text<return>` to display the values of Y-Memory starting at location 100 hexadecimal using the text display mode.

The display on your screen should be similar to the one in Fig. 1.8.

3. Type `display y:$100 -b -graph<return>` to display the values of Y-Memory starting at location 100 hexadecimal using the graphic display mode.

The display on your screen should be similar to the one in Fig. 1.9.

1.6.4 Change Command

The **change** command can be used to examine or modify the values of registers and/or memory locations.

1. Type `change x:$10 $1a<return>` to change the value of X-Memory location 10 hexadecimal to \$1A.
2. Type `display x:$10 -h -text<return>` to display the value of X-Memory location \$10 in hexadecimal using the text display mode. The value is displayed in the data window.

The display on your screen should be similar to the one in Fig. 1.10.

The screenshot shows a debugger interface with four main windows:

- [Data] [CB1M] ()**: Displays memory locations Y:0100 to Y:0105 with their corresponding hexadecimal values.
- [Unassemble] (1)**: Displays assembly instructions for addresses P:0201 to P:020A, including operations like DC, ADD, CMP, TFR, and MOVE.
- [Registers] [HEX] (1)**: Displays the current values of registers X, Y, X1, X0, Y1, Y0, A, B, and others in hexadecimal.
- [Command] [HEX] (1)**: Shows the command `DISPLAY y:$100 -b -text` and its execution status.

At the bottom, a status bar shows `Addr: A:PROG1.LOD` and other system information.

Fig. 1.8 Values of Y-Memory Locations Using the Text Display Mode

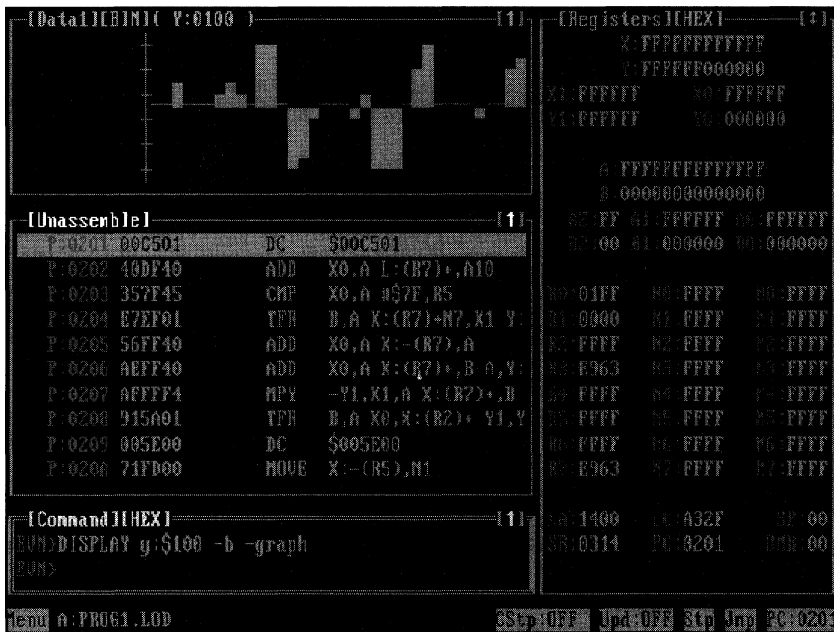


Fig. 1.9 Values of Y-Memory Locations Using the Graphic Display Mode

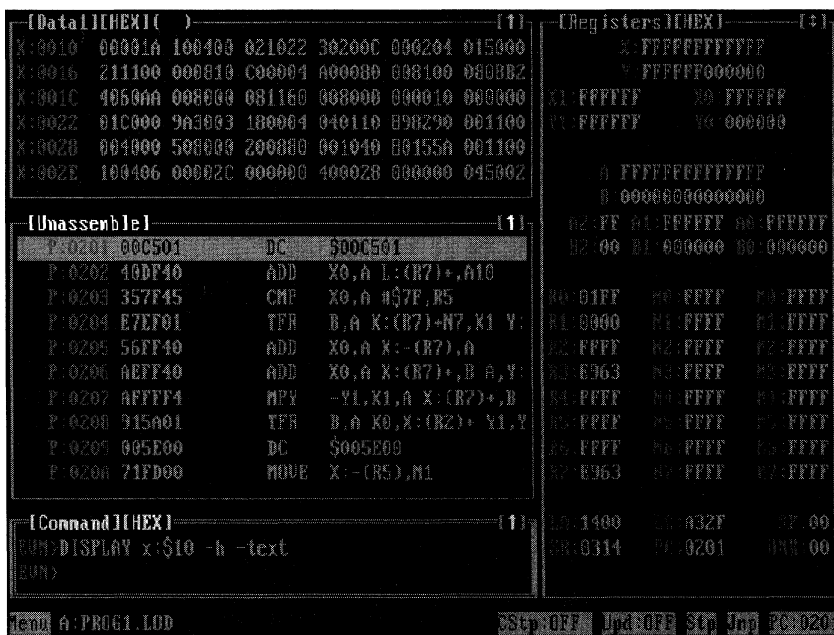


Fig. 1.10 Value of X-Memory Location 10 Hexadecimal After the Execution of the First Change Command

3. Type `change pc $10 a2 $20 b $0<return>` to change the value of the DSP56002's program counter (PC) register to \$10, and change the values of the accumulator registers A2 and B to \$20 and zero.

The display on your screen should be similar to that in Fig. 1.11.

Note that the new values of the program counter and the accumulator registers A2 and B are displayed in the registers window.

4. Type `change r0..r7 0<return>` to change the values of the DSP56002's address registers R0–R7 to 0.

The display on your screen should be similar to that in Fig. 1.12.

Note that the new values of the address registers R0..R7 are displayed in the registers window.

5. Type `change x:0#100 $1234<return>` to change the values of the X-Memory locations 0–100 to \$1234.
6. Type `display x:$0 -text<return>` to display the values of X-Memory starting at location zero using the text display mode.

The display on your screen should be similar to that in Fig. 1.13.

In order to display all the X-Memory locations in the data window, you may select the data window by moving the mouse pointer to the data window and <click-left>. A <click-left> on the “up-arrow-head” and “down-arrow-head” buttons scrolls the data window up and down by one line. A <click-left> on the upper or lower part of the shaded bar scrolls the data window by one page up and down.

The screenshot displays a debugger interface with four main windows:

- [Data][HEX] ()**: Shows memory locations from X:0010 to X:0022. For example, X:0010 contains 00001A 100400 021022 30200C 000204 015000.
- [Registers][HEX] (+)**: Shows the state of registers. X is FFFFFFFF, Y is FFFFFFF0. Accumulator A2 is 20FFFFFF and B is 0000000000000000. Address registers R0-R7 are mostly 000000.
- [Unassemble] ()**: Shows assembly instructions. For example, P:0010 is FFFFFFFF MACR -Y1,X1,B X:(R7)+,B.
- [Command][HEX] ()**: Shows the command `CHANGE pc $10 a2 $20 b $0` being entered.

At the bottom, a status bar shows `menu a:PROG1.LOD` and control flags `CStep OFF Upd OFF Stp Jmp PC:0010`.

Fig. 1.11 Values of Program Counter and Accumulator Registers After the Execution of the Second Change Command

[Data][HEX]-----[1]										[Registers][HEX]-----[1]									
X:0010	00001A	100400	021022	30200C	000204	015000				X:FFFFFF									
X:0016	211100	000810	C00004	A00000	000100	000BB2				X:FFFFFF									
X:001C	4050AA	000000	081160	000000	000010	000000				X:FFFFFF									
X:0022	01C000	9A3003	180004	040110	B38290	001100				X:FFFFFF									
X:0028	004000	500000	Z00000	001040	B01550	001100				X:FFFFFF									
X:002F	100400	00002C	000000	400028	000000	045002				X:FFFFFF									
[Unassemble]-----[1]										[Registers][HEX]-----[1]									
P:0010	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0011	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0012	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0013	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0014	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0015	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0016	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0017	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0018	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0019	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
[Command][HEX]-----[1]										[Registers][HEX]-----[1]									
CMD:CHANGE r0..r7 0										X:FFFFFF									
CMD:										X:FFFFFF									
[End] A:PROG1.LOD										X:FFFFFF									

Fig. 1.12 Values of the Address Registers After the Execution of the Third Change Command

[Data][HEX]-----[1]										[Registers][HEX]-----[1]									
X:0000	001234	001234	001234	001234	001234	001234				X:FFFFFF									
X:0008	001234	001234	001234	001234	001234	001234				X:FFFFFF									
X:0010	001234	001234	001234	001234	001234	001234				X:FFFFFF									
X:0018	001234	001234	001234	001234	001234	001234				X:FFFFFF									
X:0020	001234	001234	001234	001234	001234	001234				X:FFFFFF									
X:0028	001234	001234	001234	001234	001234	001234				X:FFFFFF									
[Unassemble]-----[1]										[Registers][HEX]-----[1]									
P:0010	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0011	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0012	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0013	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0014	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0015	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0016	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0017	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0018	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
P:0019	FFFFFF	MACR	-Y1,X1,B	X:(R7)+,B						X:FFFFFF									
[Command][HEX]-----[1]										[Registers][HEX]-----[1]									
CMD:DISPLAY x:\$0 -text										X:FFFFFF									
CMD:										X:FFFFFF									
[End] A:PROG1.LOD										X:FFFFFF									

Fig. 1.13 Values of X-Memory Locations Starting at Location Zero After the Execution of the Fourth Change Command

1.6.5 Copy Command

The **copy** command can be used to copy the values of one memory block to another.

1. Type `change x:$0..$10 $ff<return>` to change the values of X-Memory locations 0–10 hexadecimal to `$ff`.
2. Type `display x:0 -h -text <return>` to display the values of X-Memory locations starting at location zero.

The display on your screen should be similar to the one in Fig. 1.14.

3. Type `copy x:$0#$10 x:$100<return>` to copy the values of X-Memory locations 0–10 to the seventeen X-Memory locations beginning at 100 hexadecimal.
4. Type `display x:$100 -h<return>` to display the values of X-Memory locations starting at location 100 hexadecimal.

The display on your screen should be similar to the one in Fig. 1.15.

5. Type `quit<return>` to exit the Debug-EVM program.

1.7 USING THE EVM TO PROGRAM THE DSP56002

1.7.1 Enter, Assemble, and Disassemble a Canned DSP56002 Program

The Debug-EVM program has a single line, interactive assembler that accepts DSP56002 assembly language mnemonics. It can be used to enter or edit memory resident DSP56002 object code programs.

```

[Data] [HEX] ( ) [1]
X:0000 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
X:0006 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
X:000C 0000FF 0000FF 0000FF 0000FF 0000FF 001234
X:0012 001234 001234 001234 001234 001234 001234
X:0018 001234 001234 001234 001234 001234 001234
X:001E 001234 001234 001234 001234 001234 001234

[Registers] [HEX] ( ) [1]
X:FFFFFFFF
Y:FFFFFFFF000000
X1:FFFFFF X0:FFFFFF
Y1:FFFFFF Y0:000000

A:20FFFFFF
B:0000000000000000
R2:20 R1:FFFFFF R0:FFFFFF
R2:00 R1:000000 R0:000000

R0:0000 R0:FFFF R0:FFFF
R1:0000 R1:FFFF R1:FFFF
R2:0000 R2:FFFF R2:FFFF
R3:0000 R3:FFFF R3:FFFF
R4:0000 R4:FFFF R4:FFFF
R5:0000 R5:FFFF R5:FFFF
R6:0000 R6:FFFF R6:FFFF
R7:0000 R7:FFFF R7:FFFF

LA:1400 LC:032F SP:00
SR:0354 PC:0010 ONR:00

[Unassemble] ( ) [1]
P:0010 FFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0011 FFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0012 FFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0013 FFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0014 FFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0015 FFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0016 FFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0017 FFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0018 FFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0019 FFFFFFF MACR -Y1,X1,B X:(R7)+,B

[Command] [HEX] ( ) [1]
EVM: DISPLAY x:0 -h -text
EVM:

Menu A: PROG1.LOD CStop:OFF Jpd:OFF Stp Jmp PC:0010

```

Fig. 1.14 Values of X-Memory Locations Starting at Location Zero After the Execution of the Change Command

```

[Data] [HEX] (1)
X:0100 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
X:0106 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
X:010C 0000FF 0000FF 0000FF 0000FF 005B00 107E00
X:0112 043F01 E8FB00 20EE00 90DE10 BFFDD2 8C2B00
X:0118 00F300 407200 77DA01 FFFFB6 05BF40 00FF40
X:011E FA3FA9 DFF2FE 02FD01 00FA00 4BB700 55FF00

[Registers] [HEX] (1)
X:FFFFFFFF
Y:FFFFFFFF000000
X1:FFFFFF X0:FFFFFF
Y1:FFFFFF Y0:000000
0:20FFFFFF
B:00000000000000
02:20 01:FFFFFF 00:FFFFFF
02:00 01:000000 00:000000
00:0000 00:FFFF 00:FFFF
01:0000 01:FFFF 01:FFFF
02:0000 02:FFFF 02:FFFF
03:0000 03:FFFF 03:FFFF
04:0000 04:FFFF 04:FFFF
05:0000 05:FFFF 05:FFFF
06:0000 06:FFFF 06:FFFF
07:0000 07:FFFF 07:FFFF
LA:1400 LC:032F SP:00
SR:0354 PC:0010 DMM:00

[Unassemble] (1)
P:0010 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0011 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0012 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0013 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0014 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0015 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0016 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0017 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0018 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0019 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B

[Command] [HEX] (1)
EUN>DISPLAY x:$100 -h
EUN>

Menu A:PROG1.LOD 33:100FF Upd:0FF 307 JmV PG:0010

```

Fig. 1.15 Values of X-Memory Locations Starting at Location 100 Hexadecimal After the Execution of the Copy Command

1. Type assemble p:\$217<return> to begin the assembly process at P-Memory location \$217.
2. Type:


```

move x:0300,b<return>
move b,y:(r5)<return>
clr b x:(r0)+,x0<return>
move y:(r5)+,y0<return>
mac -y0,x0,b x:(r0)+,x0 y:(r5)+,y0<return>
mac y0,x0,b y:(r5),y0<return>
mac -y0,x0,b x:(r0)+,x0 y:(r4)+,y0<return>
macr y0,x0,b<return>
move b,y:(r4)-<return>
move b,x:301<return>

```

You have just entered your first DSP56002 program into the P-Memory. For now, don't concern yourself with the program itself; the DSP56002's instruction set will be covered in later chapters.

3. Select the command window and type disassemble p:\$217..\$222<return> to disassemble P-Memory locations \$217–\$222 and display the resulting DSP56002 assembly language mnemonics.

Note that this is the same program that you entered in step 2. The display on your screen should be like that in Fig. 1.16.

In order to display all the program memory locations in the unassemble

1. Type `break #1 p:$219 <return>` to show the values of the registers in the registers window and the specified memory locations in the data window that occur when the DSP56002 program instruction at P-Memory location \$219 is reached during program execution.

The register and memory location values are those that occur at the completion of the program instruction preceding the one at P-Memory location \$219.

The display on your screen should be similar to the one in Fig. 1.18.

2. Type `break #2 p:$220<return>` to show the values of the registers in the registers window and the specified memory locations in the data window that occur when the DSP56002 program instruction at P-Memory location \$220 is reached during program execution.

The register and memory location values are those that occur at the completion of the program instruction preceding the one at P-Memory location \$220.

The display on your screen should be similar to that in Fig. 1.19.

3. Type `break<return>` to display the breakpoints that are enabled.

The display on your screen should be like that in Fig. 1.20.

4. Hit `<escape>` to return to the command window.

5. Type `go $217<return>` to start program execution at P-Memory location \$217.

The values of the registers in the registers window and the specified memory locations in the data window are updated when the breakpoint location \$219

```

[Data] [HEX] ( ) [1]
X:0000 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
X:0006 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
X:000C 0000FF 0000FF 0000FF 0000FF 0000FF 001234
X:0012 001234 001234 001234 001234 001234 001234
X:0018 001234 001234 001234 001234 001234 001234
X:001E 001234 001234 001234 001234 001234 001234

[Registers] [HEX] ( ) [1]
X:FFFFFFFF
Y:FFFFFFFF000000
X1:FFFFFF X0:FFFFFF
Y1:FFFFFF Y0:000000
A:20FFFFFFFF
B:00000000000000
R2:20 R1:FFFFFF R0:FFFFFF
R3:00 R4:000000 R5:000000
R6:0000 R0:FFFF R1:FFFF
R2:0000 R3:FFFF R4:FFFF
R5:0000 R6:FFFF R7:FFFF
R8:0000 R9:FFFF R10:FFFF
R11:0000 R12:FFFF R13:FFFF
R14:0000 R15:FFFF R16:FFFF
R17:0000 R18:FFFF R19:FFFF
LA:1400 LC:A32F SP:00
SR:0354 PC:0201 DNR:00

[Unassemble] ( ) [1]
P:0217 57F000 00012C MOVE X:>$012C,B
SW P:0219 5F6500 MOVE B,Y:(R5) #1
P:021A 44D01B CLR B X:(R0)+,X0
P:021B 4EDD00 MOVE Y:(R5)+,Y0
P:021C F0B0DE MAC -Y0,X0,B X:(R0)+,X0
P:021D 4EE5DA MAC +Y0,X0,B Y:(R5),Y0
P:021E F0B0DE MAC -Y0,X0,B X:(R0)+,X0
P:021F 2000DB MACR +Y0,X0,B
P:0220 5F5400 MOVE B,Y:(R4)-
P:0221 577000 00012D MOVE B,X:>$012D

[Command] [HEX] ( ) [1]
EVM>BREAK #1 p:$219
EVM>

Menu A:PROG1.LOD CStp:OFF Upd:OFF Stp Jmp PC:0201

```

Fig. 1.18 Values of Registers and Memory Locations When the Program Instruction at P-Memory Location \$219 Is Reached

is reached during program execution. The display on your screen should be similar to that shown in Fig. 1.21.

6. Hit <f10> to continue program execution at P-Memory location \$219.

The values of the registers in the registers window and the specified memory locations in the data window are updated when the breakpoint location \$220 is reached during program execution. The display on your screen should be similar to that shown in Fig. 1.22.

7. Type `break #2 off<return>` to cancel breakpoint #2.
8. Type `go $217 #1<return>` to start program execution at P-Memory location \$217, display the registers in the registers window and the specified memory locations in the data window when breakpoint #1 is reached, and stop program execution when breakpoint #1 is reached.

The display on your screen should be similar to the one in Fig. 1.23.

9. Type `break #1 off<return>` to cancel breakpoint #1.
10. Type `change pc $217<return>` to place the value \$217 into the DSP56002's program counter (PC).

The pc now points to P-Memory location \$217.

11. Type `trace<return>` to execute one instruction and show the values of the registers in the registers window and the specified memory locations in the data window.

The display on your screen should be similar to that in Fig. 1.24.

[Data] [HEX] () [1]										[Registers] [HEX] () [1]									
X:0000	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	X:FFFFFFFF									
X:0006	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	Y:FFFFFFFF000000									
X:000C	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	001234	X1:FFFFFF	X0:FFFFFF								
X:0012	001234	001234	001234	001234	001234	001234	001234	001234	001234	Y1:FFFFFF	Y0:000000								
X:0018	001234	001234	001234	001234	001234	001234	001234	001234	001234										
X:001E	001234	001234	001234	001234	001234	001234	001234	001234	001234										
[Unassemble] [1]										A:20FFFFFFF									
P:0217	57F000	00012C	MOVE	X:>\$012C,B						B:FFA0FE10000000									
SW P:0219	576500		MOVE	B,Y:(R5)															
P:021A	44D81B		CLH	B X:(R0)+,X0						R0:0000	R0:FFFF	R0:FFFF							
P:021B	4EDD00		MOVE	Y:(R5)+,Y0						R1:0000	R1:FFFF	R1:FFFF							
P:021C	F0B8DE		MAC	-Y0,X0,B X:(R0)+,X0						R2:0000	R2:FFFF	R2:FFFF							
P:021D	4EE5DA		MAC	+Y0,X0,B Y:(R5),Y0						R3:0000	R3:FFFF	R3:FFFF							
P:021E	F098DE		MAC	-Y0,X0,B X:(R0)+,X0						R4:0000	R4:FFFF	R4:FFFF							
P:021F	2000DB		MACR	+Y0,X0,B						R5:0000	R5:FFFF	R5:FFFF							
SW P:0220	5F5400		MOVE	B,Y:(R4)-						R6:0000	R6:FFFF	R6:FFFF							
P:0221	577000	00012D	MOVE	B,X:>\$012D						R7:0000	R7:FFFF	R7:FFFF							
[Command] [HEX] [1]										LA:1400									
EUN>GO \$217										SR:03D4									
EUN>										PC:0219									
Menu A:PROG1.LOD										CStop:OFF Upd:OFF Stop:Imp PC:0219									

Fig. 1.21 Values of Registers and Memory Locations When the Breakpoint Location \$219 Is Reached

[Data1][HEX] () [1]										[Registers][HEX] () [1]									
X:0000	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	X:FFFFFF0000FF									
X:0006	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	Y:FFFFFFA0FB10									
X:000C	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	001234	X1:FFFFFF	X0:0000FF								
X:0012	001234	001234	001234	001234	001234	001234	001234	001234	001234	Y1:FFFFFF	Y0:A0FB10								
X:0018	001234	001234	001234	001234	001234	001234	001234	001234	001234										
X:001E	001234	001234	001234	001234	001234	001234	001234	001234	001234										
[Unassemble] (1)																			
P:0217	57F000	00012C	MOVE	X:>\$012C,B						R0:0003	R0:FFFF	R0:FFFF							
SW P:0219	5F6500		MOVE	B,Y:(R5)	#1					R1:0000	N1:FFFF	N1:FFFF							
P:021A	44D81B		CLR	B X:(R0)+,X0						R2:0000	N2:FFFF	N2:FFFF							
P:021B	4EDD00		MOVE	Y:(R5)+,Y0						R3:0000	N3:FFFF	N3:FFFF							
P:021C	F0B8DE		MAC	-Y0,X0,B X:(R0)+,X0						R4:0001	N4:FFFF	N4:FFFF							
P:021D	4EE5DA		MAC	+Y0,X0,B Y:(R5),Y0						R5:0002	N5:FFFF	N5:FFFF							
P:021E	F0B8DE		MAC	-Y0,X0,B X:(R0)+,X0						R6:0000	N6:FFFF	N6:FFFF							
P:021F	2000DB		MACR	+Y0,X0,B						R7:0000	N7:FFFF	N7:FFFF							
SW P:0220	5F5400		MOVE	B,Y:(R4)-	#2														
P:0221	577000	00012D	MOVE	B,X:>\$012D															
[Command][HEX] (1)																			
EUN>GO \$217										LA:1400	LC:A32F	SP:00							
EUN>										SR:03D0	PC:0220	DNR:00							
[end] A:PROG1.LOD										CStp:OFF Upd:OFF Stp:Jmp PC:0220									

Fig. 1.22 Values of Registers and Memory Locations When the Breakpoint Location \$220 Is Reached

[Data1][HEX] () [1]										[Registers][HEX] () [1]									
X:0000	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	X:FFFFFF0000FF									
X:0006	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	Y:FFFFFFA0FB10									
X:000C	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	001234	X1:FFFFFF	X0:0000FF								
X:0012	001234	001234	001234	001234	001234	001234	001234	001234	001234	Y1:FFFFFF	Y0:A0FB10								
X:0018	001234	001234	001234	001234	001234	001234	001234	001234	001234										
X:001E	001234	001234	001234	001234	001234	001234	001234	001234	001234										
[Unassemble] (1)																			
P:0217	57F000	00012C	MOVE	X:>\$012C,B						R0:0003	R0:FFFF	R0:FFFF							
SW P:0219	5F6500		MOVE	B,Y:(R5)	#1					R1:0000	N1:FFFF	N1:FFFF							
P:021A	44D81B		CLR	B X:(R0)+,X0						R2:0000	N2:FFFF	N2:FFFF							
P:021B	4EDD00		MOVE	Y:(R5)+,Y0						R3:0000	N3:FFFF	N3:FFFF							
P:021C	F0B8DE		MAC	-Y0,X0,B X:(R0)+,X0						R4:0001	N4:FFFF	N4:FFFF							
P:021D	4EE5DA		MAC	+Y0,X0,B Y:(R5),Y0						R5:0002	N5:FFFF	N5:FFFF							
P:021E	F0B8DE		MAC	-Y0,X0,B X:(R0)+,X0						R6:0000	N6:FFFF	N6:FFFF							
P:021F	2000DB		MACR	+Y0,X0,B						R7:0000	N7:FFFF	N7:FFFF							
P:0220	5F5400		MOVE	B,Y:(R4)-															
P:0221	577000	00012D	MOVE	B,X:>\$012D															
[Command][HEX] (1)																			
EUN>GO \$217 #1										LA:1400	LC:A32F	SP:00							
EUN>										SR:03D0	PC:0219	DNR:00							
[end] A:PROG1.LOD										CStp:OFF Upd:OFF Stp:Jmp PC:0219									

Fig. 1.23 Values of Registers and Memory Locations When the Breakpoint #1 Is Reached

[Data][HEX] () [1]										[Registers][HEX] [1]									
X:0000	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	X:FFFFFF0000FF									
X:0006	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	Y:FFFFFFA0FB10									
X:000C	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	001234	001234	X1:FFFFFF	X0:0000FF								
X:0012	001234	001234	001234	001234	001234	001234	001234	001234	001234	Y1:FFFFFF	Y0:A0FB10								
X:0018	001234	001234	001234	001234	001234	001234	001234	001234	001234										
X:001E	001234	001234	001234	001234	001234	001234	001234	001234	001234										
[Unassemble] [1]																			
P:021F	57F000	00012C	MOVE	X:>\$012C,B						R0:0000	R0:FFFF	R0:FFFF							
P:0219	5F6500		MOVE	B,Y:(R5)						R1:0000	R1:FFFF	R1:FFFF							
P:021A	44D81B		CLR	B X:(R0)+,X0						R2:0000	R2:FFFF	R2:FFFF							
P:021B	4EDD00		MOVE	Y:(R5)+,Y0						R3:0000	R3:FFFF	R3:FFFF							
P:021C	F0B8DE		MAC	-Y0,X0,B X:(R0)+,X0						R4:0001	R4:FFFF	R4:FFFF							
P:021D	4EE5DA		MAC	+Y0,X0,B Y:(R5),Y0						R5:0002	R5:FFFF	R5:FFFF							
P:021E	F0B8DE		MAC	-Y0,X0,B X:(R0)+,X0						R6:0000	R6:FFFF	R6:FFFF							
P:021F	2000DB		MACR	+Y0,X0,B						R7:0000	R7:FFFF	R7:FFFF							
P:0220	5F5400		MOVE	B,Y:(R4)-						LA:1400	LC:A32F	SP:00							
P:0221	577000	00012D	MOVE	B,X:>\$012D						SR:03D0	PC:0219	DNR:00							
[Command][HEX] [1]																			
EUN>TRACE																			
EUN>																			
Menu A:PROG1.LOD										CStep:OFF Upd:OFF Step Jmp PC:0219									

Fig. 1.24 Values of Registers and Memory Locations After the Execution of the First Trace Command

[Data][HEX] () [1]										[Registers][HEX] [1]									
X:0000	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	X:FFFFFF0000FF									
X:0006	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	Y:FFFFFFA0FB10									
X:000C	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	0000FF	001234	001234	X1:FFFFFF	X0:0000FF								
X:0012	001234	001234	001234	001234	001234	001234	001234	001234	001234	Y1:FFFFFF	Y0:A0FB10								
X:0018	001234	001234	001234	001234	001234	001234	001234	001234	001234										
X:001E	001234	001234	001234	001234	001234	001234	001234	001234	001234										
[Unassemble] [1]																			
P:021F	57F000	00012C	MOVE	X:>\$012C,B						R0:0000	R0:FFFF	R0:FFFF							
P:0219	5F6500		MOVE	B,Y:(R5)						R1:0000	R1:FFFF	R1:FFFF							
P:021A	44D81B		CLR	B X:(R0)+,X0						R2:0000	R2:FFFF	R2:FFFF							
P:021B	4EDD00		MOVE	Y:(R5)+,Y0						R3:0000	R3:FFFF	R3:FFFF							
P:021C	F0B8DE		MAC	-Y0,X0,B X:(R0)+,X0						R4:0001	R4:FFFF	R4:FFFF							
P:021D	4EE5DA		MAC	+Y0,X0,B Y:(R5),Y0						R5:0003	R5:FFFF	R5:FFFF							
P:021E	F0B8DE		MAC	-Y0,X0,B X:(R0)+,X0						R6:0000	R6:FFFF	R6:FFFF							
P:021F	2000DB		MACR	+Y0,X0,B						R7:0000	R7:FFFF	R7:FFFF							
P:0220	5F5400		MOVE	B,Y:(R4)-						LA:1400	LC:A32F	SP:00							
P:0221	577000	00012D	MOVE	B,X:>\$012D						SR:03D4	PC:021C	DNR:00							
[Command][HEX] [1]																			
EUN>TRACE 3																			
EUN>																			
Menu A:PROG1.LOD										CStep:OFF Upd:OFF Step Jmp PC:021C									

Fig. 1.25 Values of Registers and Memory Locations After the Execution of the Second Trace Command

12. Type `trace 3<return>` to execute the next three instructions and display the values of the register in the registers window and the specified memory locations in the data window that occur from execution of these instructions.

The display on your screen should be similar to that in Fig. 1.25.

13. Type `quit<return>` to exit the Debug-EVM program.

1.8 REFERENCE

1. Domain Technologies, *Debug-EVM User's Guide*, 1994.

Introduction to Motorola's DSP Assembler Program

The Motorola DSP Assemblers are programs that process assembly language source statements written for Motorola's family of digital signal processors. The assembler translates these source statements into object programs compatible with other Motorola DSP software and hardware products.

In this chapter, the DSP Assembler Software program and the IBM PC Line Editor (EDLIN) are used to develop canned DSP56002 programs and macros. The program development process begins by using EDLIN to enter the DSP56002 assembly language source statements of the canned DSP56002 programs and macros to produce an assembler source statement file for each program or macro. Next, the ASM56000 Assembler program is used to translate the file of assembler source statements for each program or macro into an absolute object file which can be loaded directly into the EVM for execution and testing. Figure 2.1 shows the DSP56002 program development process.

2.1 INSTALLING THE ASM56000 ASSEMBLER SOFTWARE PROGRAM

For simplicity, it is assumed that Drive "C" is assigned to the hard disk and that DOS resides in the directory c:\dos.

1. Boot the PC.

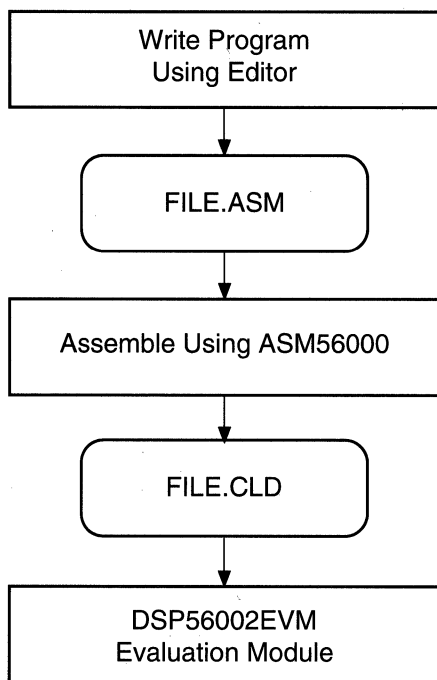


Fig. 2.1 DSP56002 Program Development Process

2. Place the diskette containing the ASM56000 Assembler Software program into Drive "A."
3. Type `copy a:\asm56000.exe c:\evm\asm56000.exe<return>` to copy the `asm56000.exe` executable program on Drive "A" to the specified directory on Drive "C."
4. Type `cd c:\evm<return>` to enter this directory.
5. Type `asm56000<return>`.

Do not concern yourself with the details displayed on your screen. This step is performed only to show you that you have successfully invoked the ASM56000 Assembler Software program. The display on your screen should be similar to that in Fig. 2.2.

2.2 PROGRAM SOURCE STATEMENT FORMAT

The ASM56000 Assembler Software program allows program source statements to use up to six fields. The fields are separated (delimited) by one or more spaces or tabs. The six fields are:

1. Label Field

The label field appears as the first field of a source statement and can take one of the following forms:

```

C:\EVM>asm56000
Motorola DSP56000 Assembler Version 5.3.2
Copyright Motorola, Inc. 1987-1994. All rights reserved.
Usage: ASM56000 [-a] [-b<objfil>] [-d<symbol> <string>] [-e<ofu> <errfil>] [-f<argfil>] [-g] [-i<ipath>] [-l<lstfil>] [-m<spath>] [-o<opt>[,<opt>...]] [-p<prdef>] [-q] [-r<rev>] [-v] [-z] [<srfil>]...
where:
-a absolute mode                      -l listing file
-b object file                        <lstfil> listing file name
  <objfil> object file name          -m macro library
-d define symbol                      <spath> library path name
  <symbol> symbol name               -o assembler option
  <string> value of symbol            <opt> option argument
-ea append to error file              -p processor revision
  <errfil> error file name            <prdef> processor designation
-eu write to error file               -q suppress banner
  <errfil> error file name            -r suppress revision
-f command file                      <rev> revision designation
  <argfil> command file name          -v verbose mode
-g debug mode                        -z strip absolute object
-i include path                      <srfil> assembler source file(s)
  <ipath> include path name
Assembler command environment variable: DSPASMOPT
C:\EVM>

```

Fig. 2.2 Invoking the ASM56000 Assembler Software Program

- a. A space or tab used as the first character of the Label Field indicates that the field is empty, and the source statement has no label.

Example: <TAB>CLR<TAB>B<RETURN>

- b. An alphabetic character used as the first character of the Label Field indicates that the source statement contains a symbol called a label.

Example: LOOP<TAB>MOVE<TAB>B,L:- (R0)<RETURN>

- c. An underscore (_) used as the first character of the Label Field indicates that the label is "local."

Example: _ENDP<RETURN>

2. Operation Field

The Operation Field appears after the Label Field and must be preceded by at least one space or tab. Entries in the Operation Field may be one of three types:

- a. **Opcode:** DSP56002 instruction mnemonic that is recognized by the ASM56000 Assembler.

Example: ENTRY<TAB>ADD B,A<RETURN>

- b. **Directive:** Special opcode that is recognized by the ASM56000 Assembler and is used to control the assembly process itself.

Example: CNST<TAB>EQU<TAB>\$5<RETURN>

- c. **Macro Call:** Invokes a previously defined macro to be inserted in place of the macro call.

3. Operand Field

The interpretation of the Operand Field is dependent on the contents of the Operation Field. The Operand Field, if present, must follow the Operation Field, and must be preceded by at least one space or tab. The Operand Field may contain a symbol, an expression, or a combination of symbols and expressions separated by commas.

Example: ENTRY<TAB>ADD B,A<RETURN>

4. Data Transfer Fields #1 and #2:

Most DSP56002 Data ALU opcodes can be appended to allow one or two data transfer operations to occur concurrently with the execution of the opcode itself. If used, Data Transfer Field #1 specifies one data transfer operation; if used, Data Transfer Field #2 specifies a second data transfer operation. A data transfer operation is specified by two addressing mode operands separated by a comma, with no embedded blanks. A Data Transfer Field must be separated (delimited) from the preceding field by at least one space or tab. The following example shows specification of one such data transfer:

Example: LOOP<TAB>MOVE<TAB>Y:-(R0),Y0<RETURN>

The following example shows specification of two such data transfers using Data Transfer Fields #1 and #2:

Example: <TAB>RND<TAB>A<TAB>X:(R0)+,X0<TAB>Y:(R4)+,Y0 <RETURN>

5. Comment Field

Comments are not considered significant to the ASM56000 Assembler Software program but can be included in the source statement for documentation purposes. A Comment Field is composed of any characters that are preceded by a semicolon (;). The semicolon can be the first character of the source statement. In this case the Comment Field becomes the entire source statement. The Comment Field can also be preceded by a space and inserted after the last of the source statement fields that are significant to the assembler program.

Examples:

```
;PROGRAM TO CLEAR THE RAM IN X and Y MEMORY <RETURN>
<TAB>CLR<TAB>B<TAB>;USED TO CLEAR LONG MEMORY <RETURN>
```

2.3 ENTERING AND EDITING A CANNED DSP56002 PROGRAM

DSP56002 programs can be entered and edited by using any available word processor program. Here, the DOS line editor **EDLIN** is used to enter, alter, display, list, load, and save a canned DSP56002 program.

1. Boot the PC.
2. Place a formatted diskette into drive "a" and type `c:\dos\edlin a:\prog2.asm<return>` to place the filename `prog2.asm` onto the diskette in Drive "A."

The message and prompt shown below should be on your screen.

New file
*

3. Type I<return> to enter the “insert mode” of the EDLIN program.
4. Type the following source statements for prog2.asm.

```
;PROGRAM TO CLEAR THE RAM IN X- AND Y-MEMORY <RETURN>
<TAB>MOVE<TAB>#$100,R0<TAB>;POINTER TO LONG MEMORY <RETURN>
<TAB>CLR<TAB>B<TAB>;USED TO CLEAR LONG MEMORY <RETURN>
LOOP<TAB>MOVE<TAB>B,L:-(R0)<TAB>;CLEAR ONE LOCATION AND DEC.<RETURN>
<TAB>MOVE<TAB>R0,A<TAB>;IS POINTER IS ZERO?<RETURN>
<TAB>TST<TAB>A<TAB>;TEST POINTER<RETURN>
<TAB>JNE<TAB>LOOP<TAB>;LOOP IF NOT DONE<RETURN>
^Z<RETURN>
```

You have just typed your second DSP56002 program. The **prog2.asm** clears the DSP56002's on chip X-Data and Y-Data memories. For now, don't concern yourself with the program itself; the DSP56002's instruction set will be covered in later chapters.

5. Type L<return> to list prog2.asm on your screen.

The prog2.asm listing on your screen should be as shown below. Verify that you have entered prog2.asm correctly. If you find an error, you can “delete” or “insert” a source statement line by typing the line number followed by a “D” (delete) or “I” (insert) command.

```
1:*;PROGRAM TO CLEAR THE RAM IN X AND Y MEMORY
2:*<TAB>MOVE<TAB>#$100,R0<TAB>;POINTER TO LONG MEMORY
3:*<TAB>CLR<TAB>B<TAB>;USED TO CLEAR LONG MEMORY
4:*<TAB>MOVE<TAB>B,L:-(R0)<TAB>;CLEAR ONE LOCATION AND DEC.
5:*<TAB>MOVE<TAB>R0,A<TAB>;IS POINTER IS ZERO?
6:*<TAB>TST<TAB>A<TAB>;TEST POINTER
7:*<TAB>JNE<TAB>LOOP<TAB>;LOOP IF NOT DONE
8:^^Z<RETURN>
```

6. Type E<return> to save prog2.asm at the destination named in step 2.

A copy of prog2.asm can be found on the DSP56002 Demonstration Software #1 diskette. For more information on the EDLIN editor refer to your DOS reference manual.

2.4 ASSEMBLING A CANNED DSP56002 PROGRAM

The ASM56000 Assembler Software program processes DSP56002 assembly language source statements into DSP56002 object code and produces a source statement listing.

1. Place the diskette containing prog2.asm into Drive “A.”
2. Type cd c:\evm<return> to enter the specified directory.
3. Type asm56000<return> to show the ASM56000 Assembler Software program's command options.

The display on your screen should be similar to that in Fig. 2.3.

```

C:\EVM>asm56000
Motorola DSP56000 Assembler Version 5.3.2
Copyright Motorola, Inc. 1987-1994. All rights reserved.
Usage: ASM56000 [-a] [-b[<objfil>]] [-d<symbol> <string>] [-e<a|w> <errfil>] [-f<argfil>] [-g] [-i<ipath>] [-l<lstfil>] [-n<npath>] [-o<opt>[,<opt>...]] [-p<proc>] [-q] [-r<rev>] [-v] [-z] <srcfil>...
where:
-a absolute mode                      -l listing file
-b object file                        <lstfil> listing file name
  <objfil> object file name          -n macro library
-d define symbol                     <npath> library path name
  <symbol> symbol name              -o assembler option
  <string> value of symbol          <opt> option argument
-ea append to error file             -p processor revision
  <errfil> error file name          <proc> processor designation
-eu write to error file             -q suppress banner
  <errfil> error file name          -r suppress revision
-f command file                     <rev> revision designation
  <argfil> command file name        -v verbose mode
-g debug mode                       -z strip absolute object
-i include path                     <srcfil> assembler source file(s)
  <ipath> include path name
Assembler command environment variable: DSPASMDPT
C:\EVM>

```

Fig. 2.3 ASM56000 Assembler Software Program's Command Options

Using the -A -B[<cldfil>] command options together instructs the ASM56000 Assembler Software program to operate in its absolute mode and create an absolute object (.cld) file.

4. Type `asm56000 -a -ba:prog2.cld a:prog2.asm<return>` to assemble the file `a:prog2.asm` in the absolute mode to produce the assembled absolute object file `a:prog2.cld`.

2.5 PROGRAM DEVELOPMENT AND EXECUTION USING THE DEBUG-EVM SOFTWARE PROGRAM

2.5.1 More Debug-EVM Software Program Basics

1. Enter the Debug-EVM Software program by following steps 5–6 of section 1.4.
2. Type `radix h<return>` to change the default radix in the command window to hexadecimal.

A hexadecimal constant can be specified by preceding the constant with a dollar (\$) symbol. Likewise, a decimal constant can be specified with a preceding grave accent (`) symbol; a binary constant can be specified with a percent (%) symbol. The radix command allows you to change the default radix to that of your convenience and then enter constants without the need for a preceding radix specification symbol. For example, if you need to enter a large number of hexadecimal constants, the `radix` command can be used to change the default

radix in the command window to hexadecimal. However, after doing so you need to precede decimal constants with a grave accent (`) radix specification symbol and precede binary constants with a percent (%) radix specification symbol.

3. Type `radix d<return>` to change the default radix in the command window to decimal.
4. Type `change x:0..10 $10<return>` to change the values of X-Memory locations 0–10 (\$0–\$a) to \$10.
5. Select the data1 window and change its radix to hexadecimal (HEX) with a mouse click on the radix button.
6. Type `display x:0..10<return>` to show the values (\$10) of X-Memory locations 0–10 (\$0–\$a) in the data1 window (see Fig. 2.4).
7. Change the radix in the data1 window to fractional (FRA) with a mouse click on the radix button.

The display on your screen should be the one in Fig. 2.5.

8. Type `version<return>` to display EVM's firmware version.

The display on your screen should be as shown in Fig. 2.6.

9. Type `wait<return>` to cause a pause condition.

When the `wait` command is encountered by the Debug-EVM Software program, processing of commands is halted until any key on the keyboard is typed.

10. Type `quit<return>` to exit the Debug-EVM Software program.

The screenshot displays the Debug-EVM software interface with several windows:

- [Data1][HEX] ()**: A table showing X-Memory locations 0 through 10, all containing the hexadecimal value 000010.
- [Registers][HEX] ()**: A table showing registers X, Y, X1, X0, Y1, Y0, and A through R7, all containing the hexadecimal value FFFFFFFF.
- [Unassemble] ()**: A list of assembly instructions, all of which are NOP (No Operation) instructions.
- [Command][DEC] ()**: The command window showing the command `DISPLAY x:0..10` entered.
- StatusBar**: At the bottom, it shows `Item A:PROG2.CLD` and various status indicators like `Step:OFF`, `Load:OFF`, `Stop`, `Run`, and `PC:0000`.

Fig. 2.4 Values of X-Memory Locations 0–10 Using Hexadecimal Radix

```

[Data] [FRA] ( ) [1] [Registers] [HEX] [1]
X:0000 0.0000019 0.0000019 0.0000019
X:0003 0.0000019 0.0000019 0.0000019
X:0006 0.0000019 0.0000019 0.0000019
X:0009 0.0000019 0.0000019 -0.7314450
X:000C -0.7301307 0.0000019 0.0001035
X:000F -0.9921417 0.0000000 0.0078125

[Unassemble] [1]
P:0030 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:003D FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:003E FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:003F FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0040 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0041 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0042 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0043 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0044 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0045 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B

[Command] [DECI] [1]
EUN>CHANGE pc 60
EUN>

LA:FFFF LC:FFFF SP:00
SR:0310 PC:003C DNR:00

Menu A:PR062.CLD X:000C-A10300 SStep:OFF Upd:OFF Step Jmd PC:003C

```

Fig. 2.5 Values of X-Memory Locations 0-10 Using Fractional Radix

```

[Data] [FRA] ( ) [1] [Registers] [HEX] [1]
X:0000 0.0000019 0.0000019 0.0000019
X:0003 0.0000019 0.0000019 0.0000019
X:0006 0.0000019 0.0000019 0.0000019
X:0009 0.0000019 0.0000019 -0.7314450
X:000C -0.7301307 0.0000019 0.0001035
X:000F -0.9921417 0.0000000 0.0078125

[Unassemble] [1]
P:0030 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:003D FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:003E FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:003F FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0040 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0041 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0042 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0043 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0044 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B
P:0045 FFFFFFFF MACR -Y1,X1,B X:(R7)+,B

[Command] [DECI] [1]
EUN-5>K rev. 1.00, Monitor 1.1
EUN>

LA:FFFF LC:FFFF SP:00
SR:0310 PC:003C DNR:00

Menu A:PR062.CLD SStep:OFF Upd:OFF Step Jmd PC:003C

```

Fig. 2.6 EVM's Firmware Version

2.5.2 More on Using the Debug-EVM Software Program to Load, Execute, and Test a Canned DSP56002 Program

1. Enter the Debug-EVM Software program by following steps 5–6 of section 1.4.
2. Place the diskette containing prog2.cld into Drive “A.”
3. Type `load a:prog2.cld<return>` to copy prog2.cld from the diskette in Drive “A” into the Debug-EVM Software’s program environment.

Remember, execution of prog2.cld clears the DSP56002 processor’s on-chip X-Memory and Y-Memory.

The display on your screen should be similar to that in Fig. 2.7.

4. Type `break #1 p:$4<return>` to set a software breakpoint at program memory \$4. The execution of prog2.cld is halted when the DSP56002 processor’s program counter (PC) has a value equal \$4.
5. Type `change x:$e0..$ff $10 y:$e0..$ff $10<return>` to change the values of X-Memory locations \$e0–\$ff and Y-Memory locations \$e0–\$ff to \$10.
6. Select the data1 window and change its radix to hexadecimal (HEX) with a mouse click on the radix button.
7. Select the command window and type `display x:$e0..$ff <return>` to display the values in X-Memory locations \$0e–\$ff in the data1 window.

The display on your screen should be similar to that in Fig. 2.8.

The screenshot displays the Debug-EVM software interface with several windows:

- [Data1]HEX1()**: A window showing memory addresses and their corresponding hexadecimal values. The values are mostly 000000, with some non-zero values at higher addresses.
- [Unassemble]**: A window showing assembly instructions. The instructions include:
 - `P 0000 60F400 000100 MOVF #5000100,R0`
 - `0000 200010 CLR R`
 - `loop:`
 - `0000 457800 MOVE R,L:-(R0)`
 - `0000 220E00 MOVE R0,R`
 - `0000 200003 TST R`
 - `0000 0E2003 JNE loop`
 - `0000 FFFFFFFF MACH -Y1,X1,B X:(R7)+,B`
 - `0000 FFFFFFFF MACH -Y1,X1,B X:(R7)+,B`
 - `0000 FFFFFFFF MACH -Y1,X1,B X:(R7)+,B`
- [Command]HEX1**: A window showing the command `load a:prog2` and the status `OK`.
- [Registers]HEX1**: A window showing the values of various registers, including PC, R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31, R32, R33, R34, R35, R36, R37, R38, R39, R40, R41, R42, R43, R44, R45, R46, R47, R48, R49, R50, R51, R52, R53, R54, R55, R56, R57, R58, R59, R60, R61, R62, R63, R64, R65, R66, R67, R68, R69, R70, R71, R72, R73, R74, R75, R76, R77, R78, R79, R80, R81, R82, R83, R84, R85, R86, R87, R88, R89, R90, R91, R92, R93, R94, R95, R96, R97, R98, R99, R100, R101, R102, R103, R104, R105, R106, R107, R108, R109, R110, R111, R112, R113, R114, R115, R116, R117, R118, R119, R120, R121, R122, R123, R124, R125, R126, R127, R128, R129, R130, R131, R132, R133, R134, R135, R136, R137, R138, R139, R140, R141, R142, R143, R144, R145, R146, R147, R148, R149, R150, R151, R152, R153, R154, R155, R156, R157, R158, R159, R160, R161, R162, R163, R164, R165, R166, R167, R168, R169, R170, R171, R172, R173, R174, R175, R176, R177, R178, R179, R180, R181, R182, R183, R184, R185, R186, R187, R188, R189, R190, R191, R192, R193, R194, R195, R196, R197, R198, R199, R200, R201, R202, R203, R204, R205, R206, R207, R208, R209, R210, R211, R212, R213, R214, R215, R216, R217, R218, R219, R220, R221, R222, R223, R224, R225, R226, R227, R228, R229, R230, R231, R232, R233, R234, R235, R236, R237, R238, R239, R240, R241, R242, R243, R244, R245, R246, R247, R248, R249, R250, R251, R252, R253, R254, R255, R256, R257, R258, R259, R260, R261, R262, R263, R264, R265, R266, R267, R268, R269, R270, R271, R272, R273, R274, R275, R276, R277, R278, R279, R280, R281, R282, R283, R284, R285, R286, R287, R288, R289, R290, R291, R292, R293, R294, R295, R296, R297, R298, R299, R300, R301, R302, R303, R304, R305, R306, R307, R308, R309, R310, R311, R312, R313, R314, R315, R316, R317, R318, R319, R320, R321, R322, R323, R324, R325, R326, R327, R328, R329, R330, R331, R332, R333, R334, R335, R336, R337, R338, R339, R340, R341, R342, R343, R344, R345, R346, R347, R348, R349, R350, R351, R352, R353, R354, R355, R356, R357, R358, R359, R360, R361, R362, R363, R364, R365, R366, R367, R368, R369, R370, R371, R372, R373, R374, R375, R376, R377, R378, R379, R380, R381, R382, R383, R384, R385, R386, R387, R388, R389, R390, R391, R392, R393, R394, R395, R396, R397, R398, R399, R400, R401, R402, R403, R404, R405, R406, R407, R408, R409, R410, R411, R412, R413, R414, R415, R416, R417, R418, R419, R420, R421, R422, R423, R424, R425, R426, R427, R428, R429, R430, R431, R432, R433, R434, R435, R436, R437, R438, R439, R440, R441, R442, R443, R444, R445, R446, R447, R448, R449, R450, R451, R452, R453, R454, R455, R456, R457, R458, R459, R460, R461, R462, R463, R464, R465, R466, R467, R468, R469, R470, R471, R472, R473, R474, R475, R476, R477, R478, R479, R480, R481, R482, R483, R484, R485, R486, R487, R488, R489, R490, R491, R492, R493, R494, R495, R496, R497, R498, R499, R500, R501, R502, R503, R504, R505, R506, R507, R508, R509, R510, R511, R512, R513, R514, R515, R516, R517, R518, R519, R520, R521, R522, R523, R524, R525, R526, R527, R528, R529, R530, R531, R532, R533, R534, R535, R536, R537, R538, R539, R540, R541, R542, R543, R544, R545, R546, R547, R548, R549, R550, R551, R552, R553, R554, R555, R556, R557, R558, R559, R560, R561, R562, R563, R564, R565, R566, R567, R568, R569, R570, R571, R572, R573, R574, R575, R576, R577, R578, R579, R580, R581, R582, R583, R584, R585, R586, R587, R588, R589, R590, R591, R592, R593, R594, R595, R596, R597, R598, R599, R600, R601, R602, R603, R604, R605, R606, R607, R608, R609, R610, R611, R612, R613, R614, R615, R616, R617, R618, R619, R620, R621, R622, R623, R624, R625, R626, R627, R628, R629, R630, R631, R632, R633, R634, R635, R636, R637, R638, R639, R640, R641, R642, R643, R644, R645, R646, R647, R648, R649, R650, R651, R652, R653, R654, R655, R656, R657, R658, R659, R660, R661, R662, R663, R664, R665, R666, R667, R668, R669, R670, R671, R672, R673, R674, R675, R676, R677, R678, R679, R680, R681, R682, R683, R684, R685, R686, R687, R688, R689, R690, R691, R692, R693, R694, R695, R696, R697, R698, R699, R700, R701, R702, R703, R704, R705, R706, R707, R708, R709, R710, R711, R712, R713, R714, R715, R716, R717, R718, R719, R720, R721, R722, R723, R724, R725, R726, R727, R728, R729, R730, R731, R732, R733, R734, R735, R736, R737, R738, R739, R740, R741, R742, R743, R744, R745, R746, R747, R748, R749, R750, R751, R752, R753, R754, R755, R756, R757, R758, R759, R760, R761, R762, R763, R764, R765, R766, R767, R768, R769, R770, R771, R772, R773, R774, R775, R776, R777, R778, R779, R780, R781, R782, R783, R784, R785, R786, R787, R788, R789, R790, R791, R792, R793, R794, R795, R796, R797, R798, R799, R800, R801, R802, R803, R804, R805, R806, R807, R808, R809, R810, R811, R812, R813, R814, R815, R816, R817, R818, R819, R820, R821, R822, R823, R824, R825, R826, R827, R828, R829, R830, R831, R832, R833, R834, R835, R836, R837, R838, R839, R840, R841, R842, R843, R844, R845, R846, R847, R848, R849, R850, R851, R852, R853, R854, R855, R856, R857, R858, R859, R860, R861, R862, R863, R864, R865, R866, R867, R868, R869, R870, R871, R872, R873, R874, R875, R876, R877, R878, R879, R880, R881, R882, R883, R884, R885, R886, R887, R888, R889, R890, R891, R892, R893, R894, R895, R896, R897, R898, R899, R900, R901, R902, R903, R904, R905, R906, R907, R908, R909, R910, R911, R912, R913, R914, R915, R916, R917, R918, R919, R920, R921, R922, R923, R924, R925, R926, R927, R928, R929, R930, R931, R932, R933, R934, R935, R936, R937, R938, R939, R940, R941, R942, R943, R944, R945, R946, R947, R948, R949, R950, R951, R952, R953, R954, R955, R956, R957, R958, R959, R960, R961, R962, R963, R964, R965, R966, R967, R968, R969, R970, R971, R972, R973, R974, R975, R976, R977, R978, R979, R980, R981, R982, R983, R984, R985, R986, R987, R988, R989, R990, R991, R992, R993, R994, R995, R996, R997, R998, R999, R1000, R1001, R1002, R1003, R1004, R1005, R1006, R1007, R1008, R1009, R1010, R1011, R1012, R1013, R1014, R1015, R1016, R1017, R1018, R1019, R1020, R1021, R1022, R1023, R1024, R1025, R1026, R1027, R1028, R1029, R1030, R1031, R1032, R1033, R1034, R1035, R1036, R1037, R1038, R1039, R1040, R1041, R1042, R1043, R1044, R1045, R1046, R1047, R1048, R1049, R1050, R1051, R1052, R1053, R1054, R1055, R1056, R1057, R1058, R1059, R1060, R1061, R1062, R1063, R1064, R1065, R1066, R1067, R1068, R1069, R1070, R1071, R1072, R1073, R1074, R1075, R1076, R1077, R1078, R1079, R1080, R1081, R1082, R1083, R1084, R1085, R1086, R1087, R1088, R1089, R1090, R1091, R1092, R1093, R1094, R1095, R1096, R1097, R1098, R1099, R1100, R1101, R1102, R1103, R1104, R1105, R1106, R1107, R1108, R1109, R1110, R1111, R1112, R1113, R1114, R1115, R1116, R1117, R1118, R1119, R1120, R1121, R1122, R1123, R1124, R1125, R1126, R1127, R1128, R1129, R1130, R1131, R1132, R1133, R1134, R1135, R1136, R1137, R1138, R1139, R1140, R1141, R1142, R1143, R1144, R1145, R1146, R1147, R1148, R1149, R1150, R1151, R1152, R1153, R1154, R1155, R1156, R1157, R1158, R1159, R1160, R1161, R1162, R1163, R1164, R1165, R1166, R1167, R1168, R1169, R1170, R1171, R1172, R1173, R1174, R1175, R1176, R1177, R1178, R1179, R1180, R1181, R1182, R1183, R1184, R1185, R1186, R1187, R1188, R1189, R1190, R1191, R1192, R1193, R1194, R1195, R1196, R1197, R1198, R1199, R1200, R1201, R1202, R1203, R1204, R1205, R1206, R1207, R1208, R1209, R1210, R1211, R1212, R1213, R1214, R1215, R1216, R1217, R1218, R1219, R1220, R1221, R1222, R1223, R1224, R1225, R1226, R1227, R1228, R1229, R1230, R1231, R1232, R1233, R1234, R1235, R1236, R1237, R1238, R1239, R1240, R1241, R1242, R1243, R1244, R1245, R1246, R1247, R1248, R1249, R1250, R1251, R1252, R1253, R1254, R1255, R1256, R1257, R1258, R1259, R1260, R1261, R1262, R1263, R1264, R1265, R1266, R1267, R1268, R1269, R1270, R1271, R1272, R1273, R1274, R1275, R1276, R1277, R1278, R1279, R1280, R1281, R1282, R1283, R1284, R1285, R1286, R1287, R1288, R1289, R1290, R1291, R1292, R1293, R1294, R1295, R1296, R1297, R1298, R1299, R1300, R1301, R1302, R1303, R1304, R1305, R1306, R1307, R1308, R1309, R1310, R1311, R1312, R1313, R1314, R1315, R1316, R1317, R1318, R1319, R1320, R1321, R1322, R1323, R1324, R1325, R1326, R1327, R1328, R1329, R1330, R1331, R1332, R1333, R1334, R1335, R1336, R1337, R1338, R1339, R1340, R1341, R1342, R1343, R1344, R1345, R1346, R1347, R1348, R1349, R1350, R1351, R1352, R1353, R1354, R1355, R1356, R1357, R1358, R1359, R1360, R1361, R1362, R1363, R1364, R1365, R1366, R1367, R1368, R1369, R1370, R1371, R1372, R1373, R1374, R1375, R1376, R1377, R1378, R1379, R1380, R1381, R1382, R1383, R1384, R1385, R1386, R1387, R1388, R1389, R1390, R1391, R1392, R1393, R1394, R1395, R1396, R1397, R1398, R1399, R1400, R1401, R1402, R1403, R1404, R1405, R1406, R1407, R1408, R1409, R1410, R1411, R1412, R1413, R1414, R1415, R1416, R1417, R1418, R1419, R1420, R1421, R1422, R1423, R1424, R1425, R1426, R1427, R1428, R1429, R1430, R1431, R1432, R1433, R1434, R1435, R1436, R1437, R1438, R1439, R1440, R1441, R1442, R1443, R1444, R1445, R1446, R1447, R1448, R1449, R1450, R1451, R1452, R1453, R1454, R1455, R1456, R1457, R1458, R1459, R1460, R1461, R1462, R1463, R1464, R1465, R1466, R1467, R1468, R1469, R1470, R1471, R1472, R1473, R1474, R1475, R1476, R1477, R1478, R1479, R1480, R1481, R1482, R1483, R1484, R1485, R1486, R1487, R1488, R1489, R1490, R1491, R1492, R1493, R1494, R1495, R1496, R1497, R1498, R1499, R1500, R1501, R1502, R1503, R1504, R1505, R1506, R1507, R1508, R1509, R1510, R1511, R1512, R1513, R1514, R1515, R1516, R1517, R1518, R1519, R1520, R1521, R1522, R1523, R1524, R1525, R1526, R1527, R1528, R1529, R1530, R1531, R1532, R1533, R1534, R1535, R1536, R1537, R1538, R1539, R1540, R1541, R1542, R1543, R1544, R1545, R1546, R1547, R1548, R1549, R1550, R1551, R1552, R1553, R1554, R1555, R1556, R1557, R1558, R1559, R1560, R1561, R1562, R1563, R1564, R1565, R1566, R1567, R1568, R1569, R1570, R1571, R1572, R1573, R1574, R1575, R1576, R1577, R1578, R1579, R1580, R1581, R1582, R1583, R1584, R1585, R1586, R1587, R1588, R1589, R1590, R1591, R1592, R1593, R1594, R1595, R1596, R1597, R1598, R1599, R1600, R1601, R1602, R1603, R1604, R1605, R1606, R1607, R1608, R1609, R1610, R1611, R1612, R1613, R1614, R1615, R1616, R1617, R1618, R1619, R1620, R1621, R1622, R1623, R1624, R1625, R1626, R1627, R1628, R1629, R1630, R1631, R1632, R1633, R1634, R1635, R1636, R1637, R1638, R1639, R1640, R1641, R1642, R1643, R1644, R1645, R1646, R1647, R1648, R1649, R1650, R1651, R1652, R1653, R1654, R1655, R1656, R1657, R1658, R1659, R1660, R1661, R1662, R1663, R1664, R1665, R1666, R1667, R1668, R1669, R1670, R1671, R1672, R1673, R1674, R1675, R1676, R1677, R1678, R1679, R1680, R1681, R1682, R1683, R1684, R1685, R1686, R1687, R1688, R1689, R1690, R1691, R1692, R1693, R1694, R1695, R1696, R1697, R1698, R1699, R1700, R1701, R1702, R1703, R1704, R1705, R1706, R1707, R1708, R1709, R1710, R1711, R1712, R1713, R1714, R1715, R1716, R1717, R1718, R1719, R1720, R1721, R1722, R1723, R1724, R1725, R1726, R1727, R1728, R1729, R1730, R1731, R1732, R1733, R1734, R1735, R1736, R1737, R1738, R1739, R1740, R1741, R1742, R1743, R1744, R1745, R1746, R1747, R1748, R1749, R1750, R1751, R1752, R1753, R1754, R1755, R1756, R1757, R1758, R1759, R1760, R1761, R1762, R1763, R1764, R1765, R1766, R1767, R1768, R1769, R1770, R1771, R1772, R1773, R1774, R1775, R1776, R1777, R1778, R1779, R1780, R1781, R1782, R1783, R1784, R1785, R1786, R1787, R1788, R1789, R1790, R1791, R1792, R1793, R1794, R1795, R1796, R1797, R1798, R1799, R1800, R1801, R1802, R1803, R1804, R1805, R1806, R1807, R1808, R1809, R1810, R1811, R1812, R1813, R1814, R1815, R1816, R1817, R1818, R1819, R1820, R1821, R1822, R1823, R1824, R1825, R1826, R1827, R1828, R1829, R1830, R1831, R1832, R1833, R1834, R1835, R1836, R1837, R1838, R1839, R1840, R1841, R1842, R1843, R1844, R1845, R1846, R1847, R1848, R1849, R1850, R1851, R1852, R1853, R1854, R1855, R1856, R1857, R1858, R1859, R1860, R1861, R1862, R1863, R1864, R1865, R1866, R1867, R1868, R1869, R1870, R1871, R1872, R1873, R1874, R1875, R1876, R1877, R1878, R1879, R1880, R1881, R1882, R1883, R1884, R1885, R1886, R1887, R1888, R1889, R1890, R1891, R1892, R1893, R1894, R1895, R1896, R1897, R1898, R1899, R1900, R1901, R1902, R1903, R1904, R1905, R1906, R1907, R1908, R1909, R1910, R1911, R1912, R1913, R1914, R1915, R1916, R1917, R1918, R1919, R1920, R1921, R1922, R1923, R1924, R1925, R1926, R1927, R1928, R1929, R1930, R1931, R1932, R1933, R1934, R1935, R1936, R1937, R1938, R1939, R1940, R1941, R1942, R1943, R1944, R1945, R1946, R1947, R1948, R1949, R1950, R1951, R1952, R1953, R1954, R1955, R1956, R1957, R1958, R1959, R1960, R1961, R1962, R1963, R1964, R1965, R1966, R1967, R1968, R1969, R1970, R1971, R1972, R1973, R1974, R1975, R1976, R1977, R1978, R1979, R1980, R1981, R1982, R1983, R1984, R1985, R1986, R1987, R1988, R1989, R1990, R1991, R1992, R1993, R1994, R1995, R1996, R1997, R1998, R1999, R2000, R2001, R2002, R2003, R2004, R2005, R2006, R2007, R2008, R2009, R2010, R2011, R2012, R2013, R2014, R2015, R2016, R2017, R2018, R2019, R2020, R2021, R2022, R2023, R2024, R2025, R2026, R2027, R2028, R2029, R2030, R2031, R2032, R2033, R2034, R2035, R2036, R2037, R2038, R2039, R2040, R2041, R2042, R2043, R2044, R2045, R2046,

[Data]HEX1										[Registers]HEX1									
X:0000	000010	000010	000010	000010	000010	000010	000010	000010	000010	X:FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
X:0001	000010	000010	000010	000010	000010	000010	000010	000010	000010	Y:FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
X:0002	000010	000010	000010	000010	000010	000010	000010	000010	000010	X1:FFFFFF	FFFFFF	X0:FFFFFF	FFFFFF	FFFFFF	FFFFFF	FFFFFF	FFFFFF	FFFFFF	FFFFFF
X:0003	000010	000010	000010	000010	000010	000010	000010	000010	000010	Y1:FFFFFF	FFFFFF	Y0:FFFFFF	FFFFFF	FFFFFF	FFFFFF	FFFFFF	FFFFFF	FFFFFF	FFFFFF
X:0004	000010	000010	000010	000010	000010	000010	000010	000010	000010	A:000000FD000000									
X:0005	000010	000010	000010	000010	000010	000010	000010	000010	000010	B:00000000000000									
X:0006	000010	000010	000010	000010	000010	000010	000010	000010	000010	R2:00 R1:0000FD R0:000000									
X:0007	000010	000010	000010	000010	000010	000010	000010	000010	000010	R3:00 R2:0000 R1:000000									
X:0008	000010	000010	000010	000010	000010	000010	000010	000010	000010	R4:00 R3:0000 R2:000000									
X:0009	000010	000010	000010	000010	000010	000010	000010	000010	000010	R5:00 R4:0000 R3:000000									
X:000A	000010	000010	000010	000010	000010	000010	000010	000010	000010	R6:00 R5:0000 R4:000000									
X:000B	000010	000010	000010	000010	000010	000010	000010	000010	000010	R7:00 R6:0000 R5:000000									
X:000C	000010	000010	000010	000010	000010	000010	000010	000010	000010	R8:00 R7:0000 R6:000000									
X:000D	000010	000010	000010	000010	000010	000010	000010	000010	000010	R9:00 R8:0000 R7:000000									
X:000E	000010	000010	000010	000010	000010	000010	000010	000010	000010	R10:00 R9:0000 R8:000000									
X:000F	000010	000010	000010	000010	000010	000010	000010	000010	000010	R11:00 R10:0000 R9:000000									
X:0010	000010	000010	000010	000010	000010	000010	000010	000010	000010	R12:00 R11:0000 R10:000000									
X:0011	000010	000010	000010	000010	000010	000010	000010	000010	000010	R13:00 R12:0000 R11:000000									
X:0012	000010	000010	000010	000010	000010	000010	000010	000010	000010	R14:00 R13:0000 R12:000000									
X:0013	000010	000010	000010	000010	000010	000010	000010	000010	000010	R15:00 R14:0000 R13:000000									
X:0014	000010	000010	000010	000010	000010	000010	000010	000010	000010	R16:00 R15:0000 R14:000000									
X:0015	000010	000010	000010	000010	000010	000010	000010	000010	000010	R17:00 R16:0000 R15:000000									
X:0016	000010	000010	000010	000010	000010	000010	000010	000010	000010	R18:00 R17:0000 R16:000000									
X:0017	000010	000010	000010	000010	000010	000010	000010	000010	000010	R19:00 R18:0000 R17:000000									
X:0018	000010	000010	000010	000010	000010	000010	000010	000010	000010	R20:00 R19:0000 R18:000000									
X:0019	000010	000010	000010	000010	000010	000010	000010	000010	000010	R21:00 R20:0000 R19:000000									
X:001A	000010	000010	000010	000010	000010	000010	000010	000010	000010	R22:00 R21:0000 R20:000000									
X:001B	000010	000010	000010	000010	000010	000010	000010	000010	000010	R23:00 R22:0000 R21:000000									
X:001C	000010	000010	000010	000010	000010	000010	000010	000010	000010	R24:00 R23:0000 R22:000000									
X:001D	000010	000010	000010	000010	000010	000010	000010	000010	000010	R25:00 R24:0000 R23:000000									
X:001E	000010	000010	000010	000010	000010	000010	000010	000010	000010	R26:00 R25:0000 R24:000000									
X:001F	000010	000010	000010	000010	000010	000010	000010	000010	000010	R27:00 R26:0000 R25:000000									
X:0020	000010	000010	000010	000010	000010	000010	000010	000010	000010	R28:00 R27:0000 R26:000000									
X:0021	000010	000010	000010	000010	000010	000010	000010	000010	000010	R29:00 R28:0000 R27:000000									
X:0022	000010	000010	000010	000010	000010	000010	000010	000010	000010	R30:00 R29:0000 R28:000000									
X:0023	000010	000010	000010	000010	000010	000010	000010	000010	000010	R31:00 R30:0000 R29:000000									
X:0024	000010	000010	000010	000010	000010	000010	000010	000010	000010	R32:00 R31:0000 R30:000000									
X:0025	000010	000010	000010	000010	000010	000010	000010	000010	000010	R33:00 R32:0000 R31:000000									
X:0026	000010	000010	000010	000010	000010	000010	000010	000010	000010	R34:00 R33:0000 R32:000000									
X:0027	000010	000010	000010	000010	000010	000010	000010	000010	000010	R35:00 R34:0000 R33:000000									
X:0028	000010	000010	000010	000010	000010	000010	000010	000010	000010	R36:00 R35:0000 R34:000000									
X:0029	000010	000010	000010	000010	000010	000010	000010	000010	000010	R37:00 R36:0000 R35:000000									
X:002A	000010	000010	000010	000010	000010	000010	000010	000010	000010	R38:00 R37:0000 R36:000000									
X:002B	000010	000010	000010	000010	000010	000010	000010	000010	000010	R39:00 R38:0000 R37:000000									
X:002C	000010	000010	000010	000010	000010	000010	000010	000010	000010	R40:00 R39:0000 R38:000000									
X:002D	000010	000010	000010	000010	000010	000010	000010	000010	000010	R41:00 R40:0000 R39:000000									
X:002E	000010	000010	000010	000010	000010	000010	000010	000010	000010	R42:00 R41:0000 R40:000000									
X:002F	000010	000010	000010	000010	000010	000010	000010	000010	000010	R43:00 R42:0000 R41:000000									
X:0030	000010	000010	000010	000010	000010	000010	000010	000010	000010	R44:00 R43:0000 R42:000000									
X:0031	000010	000010	000010	000010	000010	000010	000010	000010	000010	R45:00 R44:0000 R43:000000									
X:0032	000010	000010	000010	000010	000010	000010	000010	000010	000010	R46:00 R45:0000 R44:000000									
X:0033	000010	000010	000010	000010	000010	000010	000010	000010	000010	R47:00 R46:0000 R45:000000									
X:0034	000010	000010	000010	000010	000010	000010	000010	000010	000010	R48:00 R47:0000 R46:000000									
X:0035	000010	000010	000010	000010	000010	000010	000010	000010	000010	R49:00 R48:0000 R47:000000									
X:0036	000010	000010	000010	000010	000010	000010	000010	000010	000010	R50:00 R49:0000 R48:000000									
X:0037	000010	000010	000010	000010	000010	000010	000010	000010	000010	R51:00 R50:0000 R49:000000									
X:0038	000010	000010	000010	000010	000010	000010	000010	000010	000010	R52:00 R51:0000 R50:000000									
X:0039	000010	000010	000010	000010	000010	000010	000010	000010	000010	R53:00 R52:0000 R51:000000									
X:003A	000010	000010	000010	000010	000010	000010	000010	000010	000010	R54:00 R53:0000 R52:000000									
X:003B	000010	000010	000010	000010	000010	000010	000010	000010	000010	R55:00 R54:0000 R53:000000									
X:003C	000010	000010	000010	000010	000010	000010	000010	000010	000010	R56:00 R55:0000 R54:000000									
X:003D	000010	000010	000010	000010	000010	000010	000010	000010	000010	R57:00 R56:0000 R55:000000									
X:003E	000010	000010	000010	000010	000010	000010	000010	000010	000010	R58:00 R57:0000 R56:000000									
X:003F	000010	000010	000010	000010	000010	000010	000010	000010	000010	R59:00 R58:0000 R57:000000									
X:0040	000010	000010	000010	000010	000010	000010	000010	000010	000010	R60:00 R59:0000 R58:000000									
X:0041	000010	000010	000010	000010	000010	000010	000010	000010	000010	R61:00 R60:0000 R59:000000									
X:0042	000010	000010	000010	000010	000010	000010	000010	000010	000010	R62:00 R61:0000 R60:000000									
X:0043	000010	000010	000010	000010	000010	000010	000010	000010	000010	R63:00 R62:0000 R61:000000									
X:0044	000010	000010	000010	000010	000010	000010	000010	000010	000010	R64:00 R63:0000 R62:000000									
X:0045	000010	000010	000010	000010	000010	000010	000010	000010	000010	R65:00 R64:0000 R63:000000									
X:0046	000010	000010	000010	000010	000010	000010	000010	000010	000010	R66:00 R65:0000 R64:000000									
X:0047	000010	000010	000010	000010	000010	000010	000010	000010	000010	R67:00 R66:0000 R65:000000									
X:0048	000010	000010	000010	000010	000010	000010	000010	000010	000010	R68:00 R67:0000 R66:000000									
X:0049	000010	000010	000010	000010	000010	000010	000010	000010	000010	R69:00 R68:0000 R67:000000									
X:004A	000010	000010	000010	000010	000010	000010	000010	000010	000010	R70:00 R69:0000 R68:000000									
X:004B	000010	000010	000010	000010	000010	000010	000010	000010	000010	R71:00 R70:0000 R69:000000									
X:004C	000010	000010	000010	000010	000010	000010	000010	000010	000010	R72:00 R71:0000 R70:000000									
X:004D	000010	000010	000010	000010	000010	000010	000010	000010	000010	R73:00 R72:0000 R71:000000									
X:004E	000010	000010	000010	000010	000010	000010	000010	000010	000010	R74:00 R73:0000 R72:000000									
X:004F	000010	000010	000010	000010	000010	000010	000010	000010	000010	R75:00 R74:0000 R73:000000									
X:0050	000010	000010	000010	000010	000010	000010	000010	000010	000010	R76:00 R75:0000 R74:000000									
X:0051	000010	000010	000010	000010	000010	000010	000010	000010	000010	R77:00 R76:0000 R75:000000									
X:0052	000010	000010	000010	000010	000010	000010	000010	000010	000010	R78:00 R77:0000 R76:000000									
X:0053	000010	000010	000010	000010	000010	000010	000010	000010	000010	R79:00 R78:0000 R77:000000									
X:0																			

[Data][HEX] ()										[Registers][HEX] ()									
Y:00E0	000010	000010	000010	000010	000010	000010	000010	000010	000010	X:FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
Y:00E6	000010	000010	000010	000010	000010	000010	000010	000010	000010	X1:FFFFFF	X0:FFFFFF	Y1:FFFFFF	Y0:FFFFFF						
Y:00EC	000010	000010	000010	000010	000010	000010	000010	000010	000010										
Y:00F2	000010	000010	000010	000010	000010	000010	000010	000010	000010										
Y:00F8	000010	000010	000010	000010	000010	000010	000010	000010	000010										
Y:00FF	000010	000010	010000	000000	201000	000000													
[Unassemble]																			
P:0000	60F400	000100	MOV	B	\$000100,R0														
P:0002	20001B		CLR	B															
Loop:																			
P:0004	497800		MOV	B,L:-(R0)															
SW P:0004	220E00		MOV	R0,A	#1														
P:0006	200003		TSI	A															
P:0008	0E2003		JNE	loop															
P:000A	FFFFFF		MAC	-Y1,X1,B X:(R2)+,B															
P:000C	FFFFFF		MAC	-Y1,X1,B X:(R2)+,B															
P:000E	FFFFFF		MAC	-Y1,X1,B X:(R2)+,B															
[Command][HEX]																			
EUN>DISP y:\$00..\$FF																			
EUN>																			
Menu A:PR062.CLD																			

Fig. 2.9 Values in Y-Memory Locations \$E0-\$FF Before Execution of the Canned DSP56002 Program

[Data][HEX] ()										[Registers][HEX] ()									
Y:00E0	000010	000010	000010	000010	000010	000010	000010	000010	000010	X:FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
Y:00E6	000010	000010	000010	000010	000010	000010	000010	000010	000010	X1:FFFFFF	X0:FFFFFF	Y1:FFFFFF	Y0:FFFFFF						
Y:00EC	000010	000010	000010	000010	000010	000010	000010	000010	000010										
Y:00F2	000010	000010	000010	000010	000010	000010	000010	000010	000010										
Y:00F8	000010	000010	000010	000010	000010	000010	000010	000010	000010										
Y:00FE	000010	000000	010000	000000	201000	000000													
[Unassemble]																			
P:0000	60F400	000100	MOV	B	\$000100,R0														
P:0002	20001B		CLR	B															
Loop:																			
P:0004	497800		MOV	R,L:-(R0)															
SW P:0004	220E00		MOV	R0,A	#1														
P:0006	200003		TSI	A															
P:0008	0E2003		JNE	loop															
P:000A	FFFFFF		MAC	-Y1,X1,B X:(R2)+,B															
P:000C	FFFFFF		MAC	-Y1,X1,B X:(R2)+,B															
P:000E	FFFFFF		MAC	-Y1,X1,B X:(R2)+,B															
[Command][HEX]																			
EUN>GO																			
EUN>																			
Menu A:PR062.CLD																			

Fig. 2.10 Values in Y-Memory Locations \$E0-\$FF When Conditions of the Breakpoint Are Met

Fig. 2.11 Values in Y-Memory Locations \$E0-\$FF When Conditions of the First Breakpoint Are Met

Fig. 2.12 Values in Y-Memory Locations \$E0-\$FF When Conditions of the First Breakpoint Are Met for the Second Time

- 14.** Type step 4<return> to execute the next four program instructions.

The display on your screen should be as shown in Fig. 2.13. Note that the value of Y-Memory locations \$fb has changed from \$10 to \$0. Check that the value of X-Memory locations \$fb has also changed from \$10 to \$0.

15. Type `trace 4<return>` to execute the next four instructions with a screen update after every instruction.

The display on your screen should be as shown in Fig. 2.14. Note that the value of Y-Memory locations \$fa has changed from \$10 to \$0. Check that the value of X-Memory locations \$fa has also changed from \$10 to \$0.

- 16.** Type `quit``<return>` to exit the Debug-EVM Software program.

2.6 DEFINING AND USING MACROS

Programs often repeat a group(s) of instructions. A macro provides a shorthand means through which a group of DSP56002 instructions can be specified by a name. Therefore when typing a program, a repeated group of instructions can be compactly specified by its macro name rather than having to retype the instructions themselves.

A macro is defined by a header, a body, and a terminator. The header assigns a name to the macro and defines dummy arguments; i.e., symbolic names that will be replaced with actual arguments when the macro is called. The body contains a group of DSP56002 source instructions. The terminator is the End Macro (ENDM) directive.

[Data] [HEX] ()										[Registers] [HEX] ()									
Y:00E0 000010 000010 000010 000010 000010 000010										X:FFFFFFFFFFFFFF									
Y:00E6 000010 000010 000010 000010 000010 000010										Y:FFFFFFFFFFFFFF									
Y:00EC 000010 000010 000010 000010 000010 000010										X1:FFFFFF X0:FFFFFF									
Y:00F2 000010 000010 000010 000010 000010 000010										Y1:FFFFFF Y0:FFFFFF									
Y:00F8 000010 000010 000010 000000 000000 000000																			
Y:00FE 000000 000000 010000 000000 201000 000000										A 000000FC000000									
										B 00000000000000									
										02:00 A1:0000FC A0:000000									
										D2:00 D1:000000 D0:000000									
[Unassemble]										[]									
P:0000 50F400 000100 MOVE #>\$000100,R0																			
P:0002 20001B CLR B																			
loop:																			
P:0003 497800 MOVE B,L:-(R0)																			
HW P:0004 220E00 MOVE R0,A #1																			
P:0005 200003 TST A																			
P:0006 0E2003 JNE loop																			
P:0007 FFFFFFF MACH -Y1,X1,B X:(R2)+,B										R0:00FB R0:FFFF R0:FFFF									
P:0008 FFFFFFF MACH -Y1,X1,B X:(R2)+,B										R1:0000 R1:FFFF R1:FFFF									
P:0009 FFFFFFF MACH -Y1,X1,B X:(R2)+,B										R2:FFFF R2:FFFF R2:FFFF									
										R3:3007 R3:FFFF R3:FFFF									
										R4:FFFF R4:FFFF R4:FFFF									
										R5:FFFF R5:FFFF R5:FFFF									
										R6:FFFF R6:FFFF R6:FFFF									
										R7:3007 R7:FFFF R7:FFFF									
[Command] [HEX]										[]									
RUN>STEP 1										LA:FFFF LC:FFFF SP:00									
RUN>										SR:0310 PC:0004 DMR:00									
Menu A:PROG2.CLD										SStp:OFF Upd:OFF Stp Jmp PC:0004									

Fig. 2.13 Values in Y-Memory Locations \$E0-\$FF After Execution of the Step Command

[Data][HEX] ()										[Registers][HEX] (+)									
Y:00E0	000010	000010	000010	000010	000010	000010	000010	000010	000010	X:FFFFFFFF									
Y:00E6	000010	000010	000010	000010	000010	000010	000010	000010	000010	Y:FFFFFFFF									
Y:00EC	000010	000010	000010	000010	000010	000010	000010	000010	000010	X1:FFFFFF	X0:FFFFFF								
Y:00F2	000010	000010	000010	000010	000010	000010	000010	000010	000010	Y1:FFFFFF	Y0:FFFFFF								
Y:00F8	000010	000010	000010	000000	000000	000000	000000	000000	000000										
Y:00FE	000000	000000	010000	000000	000000	201000	000000												
[Unassemble] (1)										A:000000FB000000									
P:0000	50F400	000100	MOVE	#000100,R0						B:00000000000000									
P:0002	20001B		CLR	B						02:00	01:0000FB	00:000000							
										02:00	01:000000	00:000000							
loop:																			
P:0003	497800		MOVE	B,L:-(R0)						R0:00FA	R0:FFFF	R0:FFFF							
P:0004	220000		MOVE	R0,A						R1:0000	R1:FFFF	R1:FFFF							
P:0005	200003		TSI	A						R2:FFFF	R2:FFFF	R2:FFFF							
P:0006	0E2003		JNE	loop						R3:3D07	R3:FFFF	R3:FFFF							
P:0007	FFFFFF		MACR	-Y1,X1,B X:(R7)+,B						R4:FFFF	R4:FFFF	R4:FFFF							
P:0008	FFFFFF		MACR	-Y1,X1,B X:(R7)+,B						R5:FFFF	R5:FFFF	R5:FFFF							
P:0009	FFFFFF		MACR	-Y1,X1,B X:(R7)+,B						R6:FFFF	R6:FFFF	R6:FFFF							
										R7:3D07	R7:FFFF	R7:FFFF							
[Command][HEX] (1)										LA:FFFF LC:FFFF SP:00									
RUN>TRACE 1										SR:0310 PC:0004 DNR:00									
RUN>																			
Menu A:PROG2.CLD										CStp:OFF Upd:OFF Stp Jmp PC:0004									

Fig. 2.14 Values in Y-Memory Locations \$E0-\$FF After Execution of the Trace Command

A macro is specified within a program by a macro call statement. The macro call statement has three fields: a Label Field, an Operation Field, and an Operand Field. The Label Field, if used, corresponds to the value of the software location counter at the beginning of the macro expansion; i.e., substitution of the related instruction group in place of the macro call statement. The Operation Field contains the name of the macro. The Operand Field contains actual arguments that will be substituted in place of the dummy arguments used in the header part of the macro's definition.

2.6.1 Programming a Canned DSP56002 Macro

DSP56002 programs can be entered and edited by using any available word processor program. Here, the DOS line editor EDLIN is used to enter, display, and save a canned DSP56002 macro.

1. Boot the PC.
2. Place a formatted diskette into Drive "A" and type `c:\dos\edlin a:\mal.asm<return>` to place the filename `mal.asm` onto the diskette in Drive "A."

The message and prompt shown below should be on your screen.

```
New file
*
```

3. Type `I<return>` to enter the "insert mode" of the EDLIN program.

4. Type `MA1<tab>MACRO<tab>VALUE<return>` to define the macro's header.

This header assigns the name `MA1` and the dummy argument `VALUE` to the macro.

5. Type:

```
;MACRO TO CLEAR THE RAM IN X- AND Y-MEMORY<return>
<tab>MOVE<tab>#VALUE,R0<tab>; POINTER TO LONG
MEMORY<return>
<tab>CLR<tab>B<tab>;USED TO CLEAR LONG MEMORY
<return>
LOOP<tab>MOVE<tab>B,L:- (R0)<tab>;CLEAR 1 LOCATION
& DEC<ret>
<tab>MOVE<tab>R0,A<tab>;DOES POINTER EQUAL ZERO?
<return>
<tab>TST<tab>A<tab>;TEST POINTER<return>
<tab>JNE<tab>LOOP<tab>;LOOP IF NOT DONE<return>
```

You have just typed the body of the macro `MA1`. `MA1` clears the DSP56002 processor's on-chip X-Memory and Y-Memory. For now, don't concern yourself with the instructions themselves. The DSP56002 processor's instruction set will be covered in later chapters.

6. Type `<tab>ENDM<return>^Z` to enter the macro's terminator and exit the insert mode of EDLIN.
7. Type `E<return>` to save `ma1.asm` at the destination named in step 2.

A copy of `ma1.asm` can be found on the DSP56002 Demonstration Software diskette.

8. Type `c:\dos\edlin a:prog3.asm<return>` to place the filename `prog3.asm` onto the diskette in Drive "A."

The following message and prompt should be on your screen:

```
New file
*_
```

9. Type `<tab>INCLUDE<tab>'MA1'<tab>VALUE<return>` to "include" the `ma1` macro into `prog3.asm`.

Assembler directives are instructions to the ASM56000 Assembler Software program itself. When writing programs, the ASM56000 Assembler Software directives can be used for storage allocation, symbol and data definition, assembly control, output listing control, object file control, macros and conditional assembly, and structured programming. One such assembler directive is named `INCLUDE` and is used as shown above. The subset of ASM56000 assembler directives that is used in this book is included in Table 2.1.

Examples of their use include:

- ✗ The assembler directive `ORG P:$100` sets the runtime memory space to the P-Memory starting at location \$100
- ✗ The assembler directive `COUNT EQU $12` assigns the value \$12 to the symbol `COUNT`

Table 2.1 Subset of ASM56000 Assembler Directives

Directive	Description
DC	Define constant
DS	Define Storage
DSM	Define modulo storage
DSR	Define reverse carry storage
DUP	Duplicate a sequence of source lines
END	End of source program
EQU	Equate symbol to a value
INCLUDE	Include secondary file
MACRO	Macro Definition
ORG	Initialize memory space and location counters
SET	Set symbol to a value

- ✗ The assembler directives `ORG X:$100` and `TABLE DC 0.1, 0.2, 0.3` store the decimal values 0.1, 0.2, and 0.3 in consecutive X-Memory locations beginning at location \$100
- ✗ The assembler directives `ORG X:$100` and `STATES DS 6` reserve for `STATES` variables six consecutive X-Memory locations beginning at location \$100

10. Type `VALUE<tab>EQU<tab>$100<return>` to assign the value \$100 to the symbol "value".

Note that the `EQU` assembler directive is used to assign to the value \$100 to the symbol `VALUE`.

11. Type `<tab>MA1<tab>VALUE<return>` to enter the macro call statement that invokes the `ma1` macro.
12. Type `<tab>END<return>^Z` to indicate the logical end of `prog3.asm`.
Note the use of the assembler `END` directive.
13. Type `E<return>` to save `prog3.asm` at the destination named in step 8.

A copy of `prog3.asm` can be found on the DSP56002 Demonstration Software diskette.

2.6.2 Assembling a Canned DSP56002 Macro

1. Boot the PC.
2. Type `cd c:\evm<return>` to enter this directory.
3. Type `asm56000 -a -ba:prog3.cld -ia: a:prog3.asm<return>` to assemble `prog3.asm` and output the assembled absolute object file `prog3.cld` to Drive "A."

The **-I** option above directs the ASM56000 Assembler Software program to search the **a:** directory for the **include** file **ma1.asm**.

2.6.3 Loading a Canned DSP56002 Macro

1. Enter the Debug-EVM Software program by following steps 5–6 of section 1.4.
2. Place the diskette containing prog3.cld into Drive “A.”
3. Type `load a:prog3.cld<return>` to load prog3.cld from the diskette in Drive “A” into the Debug-EVM Software program environment.
4. Type `disassemble p:0..10<return>` to disassemble and display the contents of P-Memory locations 0–10 (\$0–\$a).

The display on your screen should be as that shown in Fig. 2.15. Note its similarity to the display in step 3 of section 2.5.2.

5. Type `quit<return>` to exit the Debug-EVM software program.
6. For more information on writing DSP56002 assembly language programs, refer to Motorola’s *DSP56000 Macro Assembler Reference Manual*.

```

[Data] [FPR] ( ) [11] [Registers] [HEX] ( ) [11]
0000 0.0041504 0.0000000 0.0000041
0001 0.0000000 0.0021657 0.0000002
0002 0.0004882 0.0781250 0.0000021
0003 0.0166953 0.0161151 0.7314450
0004 -0.7382892 0.0000019 0.0001635
0005 -0.3921565 0.0000000 0.0075125

[Unassemble] [11]
P:0000 60F400 000100 MOVE R>S000100, R0
P:0001 20001B CLR B
LOOP:
P:0002 497800 MOVE B, L:-(R0)
P:0003 220E00 MOVE R0, A
P:0004 200003 TST A
P:0005 0E2003 JNE LOOP
P:0006 FFFFFFFF MACH -Y1, X1, B X: (R7)+, B
P:0007 FFFFFFFF MACH -Y1, X1, B X: (R7)+, B
P:0008 FFFFFFFF MACH -Y1, X1, B X: (R7)+, B
P:0009 FFFFFFFF MACH -Y1, X1, B X: (R7)+, B
P:000A 09BB B7 FFFF B6 FFFF B5 FFFF B4 FFFF B3 FFFF B2 FFFF B1 FFFF B0 FFFF
P:000B 0314 000000 000000

[Command] [DEC] [11]
DISASSEMBLE p:0..10
OK

End A:PROG3.CLD 85CpOFF Jpd407F Stop Jmp 000000

```

Fig. 2.15 Loading Prog3 Program

2.7 REFERENCES

1. Motorola Inc., *DSP56000 Digital Signal Processing Family Manual*, 1992.
2. Motorola Inc., *DSP56002 Digital Signal Processor User's Manual*, 1993.

DSP56002 Architecture and Addressing Modes

In this chapter, the DSP56002 processor's architecture, addressing modes, and some of its machine instructions are introduced. The chapter begins with a review of the DSP56002 processor's architecture. Its three concurrent processing execution units, the Data Arithmetic Logic Unit, the Program Controller, and the Address Generation Unit are then described. Three of the DSP56002 processor's addressing modes—Register Direct, Special, and Register Indirect—are also discussed. A detailed description of the DSP56002 processor's instruction set will be covered in chapter 4.

3.1 DSP56002 ARCHITECTURE REVIEW

The DSP56K family of processors is built on a standard Central Processing Unit (CPU). In the expansion area around the CPU, the chip can support various configurations of memory and peripheral modules that may change from family member to family member.

In the remaining chapters, we will focus on the DSP56002 24-bit digital signal processor, its CPU, memory, and peripheral modules. The major components of the DSP56002 processor's architecture are shown in Fig. 1.1 of chapter 1 and are listed below. The block diagram in Fig. 1.1 includes both the CPU and the expansion area for memory and peripherals.

3.1.1 DSP56K Central Processing Unit

1. **Buses:** The internal multiple-bus architecture of the DSP56002 processor consists of four bidirectional 24-bit data buses and two unidirectional and one bidirectional 16-bit address buses.
 - a. **Data Buses:** The data buses include: the X Data Bus (XDB), the Y Data Bus (YDB), the Program Data Bus (PDB), and the Global Data Bus (GDB). The X Data Bus and Y Data Bus transfer data between the Data ALU and the X Data Memory and Y Data Memory, respectively. The X Data Bus and Y Data Bus also can be concatenated, by certain instructions, to function as one 48-bit bus. The Program Data Bus transfers prefetched instruction words. The Global Data Bus handles *other* data transfers such as I/O transfers to and from peripheral devices.
 - b. **Address Buses:** The address buses include: the X Address Bus (XAB), the Y Address Bus (YAB), and the Program Address Bus (PAB). The XAB and YAB provide address data that points to specific locations in the internal X Data Memory and Y Data Memory, respectively. The PAB provides address data that points to a specific location in the internal P Program Memory. External memory spaces are addressed over a single 16-bit unidirectional address bus driven by a three-input multiplexer that can select the XAB, the YAB, or the PAB.
2. **Execution Units:** The core of the DSP56002 processor consists of three execution units that operate concurrently: the Data Arithmetic Logic Unit, the Program Controller, and the Address Generation Unit. The execution units are described in sections 3.3, 3.4, and 3.5, respectively.
3. **Memory Expansion Port (Port A):** The Memory Expansion port is composed of: a 16-bit address bus, a 24-bit bidirectional data bus, and bus control signals. It is used to interface the DSP56002 processor to a wide variety of memory and peripheral devices. These devices include high-speed static RAMs, slower memory devices, and other DSPs and MPUs in master/slave configurations.
4. **On-Chip Emulator (OnCE):** The on-chip emulator (OnCE) allows the user to interact with the DSP56002's CPU and peripherals nonintrusively to examine registers, memory, or on-chip peripherals. It provides simple, inexpensive, and speed independent access to the internal registers for debugging and economical system development.
5. **Phase-Locked Loop (PLL) Based Clocking:** The Phase-Locked Loop (PLL) allows the DSP to use the available external system clock for full-speed operation, while also supplying an output clock synchronized to a synthesized internal clock. The PLL performs frequency multiplication, skew elimination, and low-power division.

3.1.2 Expansion Area

1. **Internal Memories:** The DSP56002 processor has six on-chip memories: the X Data RAM, the X Data ROM, the Y Data RAM, the Y Data ROM, the P Program

Memory RAM, and the Bootstrap ROM. The X Data RAM and Y Data RAM are 24-bit internal memories that occupy the lowest 256 locations in the X-Memory address space and the Y-Memory address space, respectively. The X Data ROM and Y Data ROM are 24-bit internal memories which, when enabled by the Operating Mode Register, occupy the next lowest 256 locations in the X-Memory address space and Y-Memory address space, respectively. The P Program RAM is a 24-bit internal memory that occupies the lowest 512 locations in the P-Memory address space. The P Program RAM contains machine instructions, constants, and data tables that are fixed at assembly time. Unused locations in the P Program RAM, X Data RAM, and Y Data RAM can be used for general purpose storage. For example, unused locations in the X Data Memory and Y Data Memory could be used to store program overlays for the P Program Memory. The Bootstrap ROM is a 64-location by 24-bit factory programmed ROM that is used only in the bootstrap mode. The bootstrap ROM is programmed to perform the bootstrap operation from the memory expansion port (port A), from the host interface, or from the serial communication interface. It provides a convenient, low-cost method of loading the program RAM with a user program after power-on reset.

2. **On-chip Peripherals:** The DSP56002 processor's on-chip peripherals provide a microcomputer-style I/O capability. The on-chip peripherals include: a parallel host interface with DMA support, a Synchronous Serial Interface (SSI) port, a Serial Communication Interface (SCI) port, and a Programmable I/O port. The 8-bit parallel Host Interface port provides a means through which a host processor or DMA controller can access the DSP56002 processor. The SCI port generally provides a standard serial I/O interface, using asynchronous character protocols, for communication with MCUs, other DSP chips, or peripherals such as a modem. The SSI port generally provides a high-speed synchronous serial data interface for communication with other DSPs or other synchronous serial devices. The Programmable I/O port provides a general purpose interface.
3. **Timer/Event Counter:** The timer can use internal or external clocking and can interrupt the processor after a number of events (clocks) specified by a user program, or it can signal an external device after counting internal events.

3.2 HOW TO PROCEED

Load the Debug-EVM Software program by following steps 5–6 of section 1.4.

3.3 DATA ALU EXECUTION UNIT

The Data ALU execution unit, shown in Fig. 3.1, performs arithmetic and logical operations on data operands. The major components of the Data ALU include: data registers, a parallel multiply-accumulator/logic unit (MAC), an accumulator shifter, a bit manipulation unit, and two shifter/limiters.

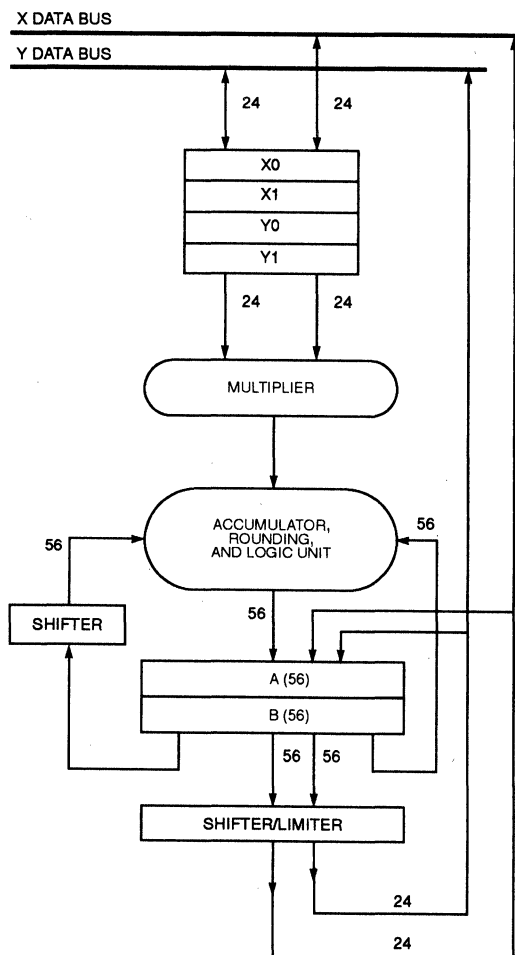


Fig. 3.1 Data ALU Block Diagram

3.3.1 Data Registers

The Data ALU has four input data registers and six data accumulator registers as shown in Fig. 3.2.

3.3.1.1 Data Input Registers: X1,X0; Y1,Y0 The data input registers can be treated as four independent 24-bit registers, X1, X0, Y1, and Y0, or as two 48-bit registers, X and Y. The 48-bit X and Y registers are developed by the concatenation of X1:X0 and Y1:Y0, respectively.

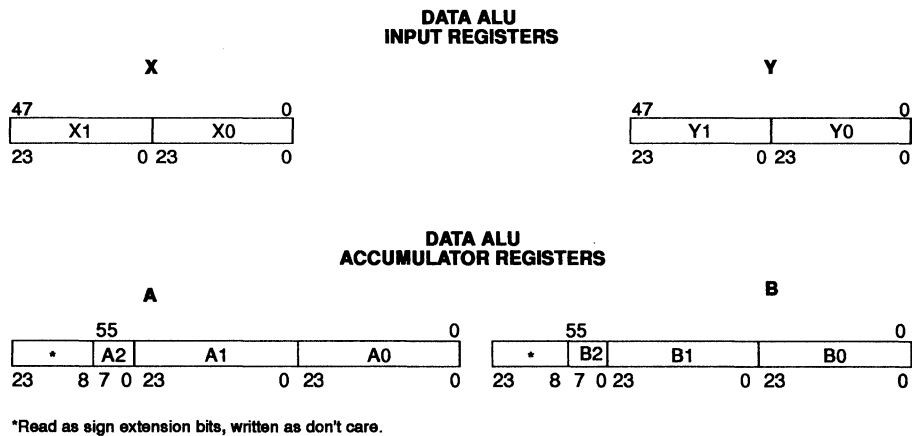


Fig. 3.2 Data Registers

E X A M P L E 3 – 1

Moving data to a 24-bit data input register

Problem

Execute the instruction **MOVE # \$111100,X1** to move the value \$111100 to the X1 Data Input Register. Assume that the contents of the X1, X0, and X data registers before execution of the instruction are:

x = \$333333333333
x1 = \$333333 x0 = \$333333

Procedure

Type: change x \$333333333333<return>
asm p:\$200 move # \$111100,x1<return>
change pc \$200<return>
step<return>

Results

x = \$111100333333
x1 = \$111100 x0 = \$333333

E X A M P L E 3 – 2

Moving data to a 48-bit data input register

Problem

Execute the instruction **MOVE L:\$100,X** to move the contents of the L-Memory location \$100 (concatenation of X-Memory and Y-Memory locations \$100) to the X Data Input Reg-

ister. Assume that the contents of the X Data Input Register and L-Memory location \$100 before execution of the instruction are:

```
l:$100 $333333444444
x = $111111222222
```

Change the radix in the Data1 Window to hexadecimal with a mouse click in the radix button.

Procedure

```
Type: change l:$100 $333333444444 x $111111222222<return>
      display l:$100 <return>
      asm p:$202 move l:$100,x<return>
      change pc $202<return>
      step<return>
```

Results

```
l:$100 $333333444444
x = $333333444444
```

3.3.1.2 Data Accumulator Registers: A2,A1,A0; B2,B1,B0 The six data accumulator registers, A2, A1, A0, B2, B1, and B0, form two general purpose 56-bit accumulators, A and B. The A-Accumulator is the concatenation of registers A2:A1:A0; the B-Accumulator is the concatenation of registers B2:B1:B0. Registers A1, A0, B1, and B0 are each 24-bits wide; registers A2 and B2 are each 8-bits wide. The two accumulators can be viewed as being 48-bits wide with 8-bit extensions. As a result, A2 and B2 are called extension registers.

E X A M P L E

3 - 3

Moving a 24-bit operand to a 56-bit accumulator

Problem

Execute the two instructions **MOVE X0,A** and **MOVE X1,B** to demonstrate that automatic sign-bit extension occurs when moving a 24-bit data operand to the 56-bit A-Accumulator and when moving a 24-bit data operand to the 56-bit B-Accumulator. Assume that the contents of the A-Accumulator and B-Accumulator registers and the X0 and X1 data input registers before execution of the two instructions are:

```
x0 = $888888
x1 = $111111
a = $00222222222222
a2 = $00 a1 = $222222 a0 = $222222
b = $00222222222222
b2 = $00 b1 = $222222 b0 = $222222
```

Procedure

```
Type: change x0 $888888 x1 $111111<return>
      change a $00222222222222<return>
```

```

change b $00222222222222<return>
asm P:$204 move x0,a<return>
asm p:$206 move x1,b<return>
change pc $204<return>
step 4<return>

```

Results

```

x0 = $888888
x1 = $111111
a = $ff888888000000
a2 = $ff  a1 = $888888  a0 = $000000
b = $00111111000000
b2 = $00  b1 = $111111  b0 = $000000

```

Note that the A2 and B2 accumulator registers are sign-extended from A1 and B1, respectively, and that the A0 and B0 accumulator registers are automatically zeroed.

E X A M P L E

3 – 4

Moving a 16-bit operand to a 56-bit accumulator*Problem*

Execute the instruction **MOVE R0,A** to move the contents of the 16-bit R0 address register to the 56-bit A-Accumulator. Assume that the contents of Address Register R0 and the A-Accumulator before execution of the instruction are:

```
r0 = $8888    a = $00111111111111
```

Procedure

```

Type: change r0 $8888 a $00111111111111<return>
asm p:$208 move r0,a<return>
change pc $208<return>
step<return>

```

Results

```
r0 = $8888    a = $00008888000000
```

Note that A-Accumulator Register A0 is zeroed and that sign-extension did not occur.

E X A M P L E

3 – 5

Moving a 56-bit A-accumulator operand to both a 16-bit destination and a 24-bit destination*Problem*

Execute the two instructions **MOVE A,R2** and **MOVE A,X1** to move the contents of the 56-bit A-Accumulator to the 16-bit R2 Address Register and the 24-bit X1 Data Input Register, respectively. Assume that the contents of the A-Accumulator, the R2 Address Register, and the X1 Data Input Register before execution of the two instructions are:


```
a = $00123456789abc
r2 = $0000
x1 = $000000
```

Procedure

```
Type: change a $00123456789abc r2 $0000<return>
      change x1 $000000<return>
      asm p:$209 move a,r2<return>
      asm p:$20a move a,x1<return>
      change pc $209<return>
      step 2<return>
```

Results

```
a = $00123456789abc
r2 = $3456
x1 = $123456
```

E X A M P L E

3 - 6

Moving a data operand to an accumulator without automatic sign-extension

Problem

Execute the instruction **MOVE # \$111111,A1** to show that by specifying the destination accumulator register, A1, A0, B1, or B0, a 24-bit data operand can be moved to an accumulator *without* automatic sign-extension and zero filling. Assume that the contents of the A, A2, A1, and A0 accumulator registers before execution of the instruction are:

```
a=$ff888888222222
a2=$ff a1=$888888 a0=$222222
```

Procedure

```
Type: change a $ff888888222222<return>
      asm p:$20b move # $111111,a1<return>
      change pc $20b<return>
      step<return>
```

Results

```
a = $ff111111222222
a2 = $ff a1 = $111111 a0 = $222222
```

3.3.2 MAC and Logic Unit

The MAC and Logic Unit performs all of the DSP56002 processor's data operand calculations such as multiplication, addition, subtraction, AND, OR, XOR, and NOT. It accepts up to three input data operands and produces one 56-bit result, where the result is always stored in either the A-Accumulator or the B-Accumulator.

E X A M P L E 3 – 7

Addition*Problem*

Execute the instruction **ADD A,B** to add the A-Accumulator to the B-Accumulator and store the result in the B-Accumulator. Assume that the contents of the A-Accumulator and the B-Accumulator before execution of the instruction are:

a = \$0011111111111111 b = \$0022222222222222

Procedure

Type: change a \$0011111111111111 b \$0022222222222222<return>
asm p:\$210 add a,b<return>
change pc \$210<return>
step<return>

Results

a=\$0011111111111111 b=\$0033333333333333

E X A M P L E 3 – 8

Subtraction*Problem*

Execute the instruction **SUB X,A** to subtract the 48-bit operand of the X Data Input Register from the A-Accumulator and store the result in the A-Accumulator. Assume that the contents of the X Data Input Register and the A-Accumulator before execution of the instruction are:

x=\$1111111111111111 a=\$0033333333333333

Procedure

Type: change x \$1111111111111111 a \$0033333333333333<return>
asm p:\$211 sub x,a<return>
change pc \$211<return>
step<return>

Results

x=\$1111111111111111 a=\$0022222222222222

E X A M P L E 3 – 9

Logical AND*Problem*

Execute the instruction **AND X1,A** to logically AND the 24-bit operand of the X1 Data Input Register with the A-Accumulator and store the result in the A-Accumulator. Assume that the contents of the X1 Data Input Register and the A-Accumulator before the execution of the instruction are:

x1=\$f0f0f0 a=\$ff888888888888

Procedure

Type: change x1 \$f0f0f0 a \$ff88888888888888<return>
 asm p:\$212 and x1,a<return>
 change pc \$212<return>
 step<return>

Results

x1=\$f0f0f0 a=\$ff808080888888

3.3.2.1 Two's-Complement Fractional Data Representation The MAC and Logic Unit uses the two's-complement fractional data representation commonly used in DSP algorithms, where a fraction is any number whose magnitude is greater than or equal to zero and less than one. Figure 3.3 shows the bit weighting of fractional data operands.

EXAMPLE 3 - 1 0

Positive two's-complement fractional data

Problem

Change the contents of the Y0 Data Register to 0.875. Hit the function key F2 to change the default radix in the command window to fractional.

Procedure

Type: change y0 0.875<return>

Results

y0 = \$7000000 = %01110000000000000000000000000000
 = $1*2^{-0} + 1*2^{-1} + 1*2^{-2} + 1*2^{-3} = 0.875$

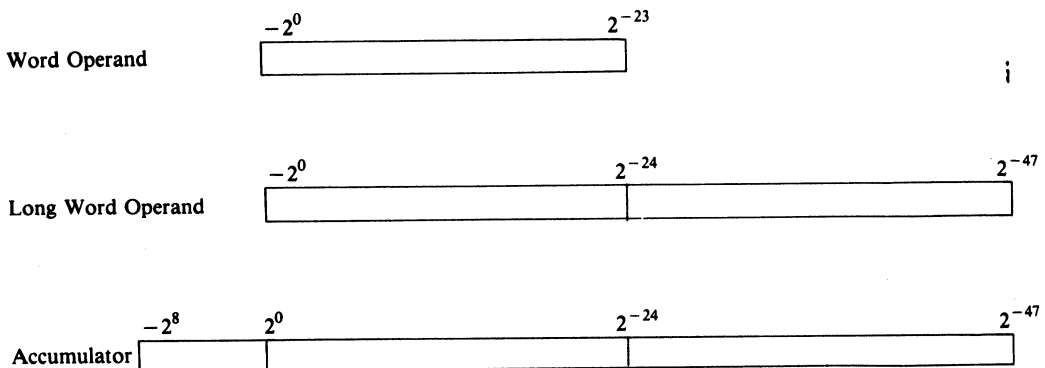


Fig. 3.3 Bit Weighting of Operands

Change the contents of the X0 Data Register to -0.875. Hit the function key F2 to change the default radix in the command window to fractional.

```
Type: change x0 -0.875<return>
```

Results

```
x0 = $90000000 = %10010000000000000000000000000000  
= -0.875
```

To convert the above two's-complement binary number 100100...00 to decimal, each bit of the number must be complemented to form the one's-complement number 011011...11. Then a one is added to the one's-complement number to form the binary number 011100..00, which is equal to the decimal number 0.875. The sign of the decimal number 0.875 is negative because the sign-bit of its equivalent two's-complement binary number is a one.

3.3.2.2 Rounding The MAC and Logic Unit can convergently round the least significant portion of an accumulator, A0 or B0, into the most significant portion, A1 or B1.

Execute the instruction **RND A** to show that a one is added to A-Accumulator Register A1 when the value of A-Accumulator Register A0 is greater than 0.5 (\$800000). Assume that the contents of the A1 and A0 accumulator registers and the SR Status Register before execution of the instruction are:

```
sr=$00c0
a1=$111111    a0=$808080
```

Procedure

```
Type: change a1 $111111 a0 $808080<return>
      change sr $00c0<return>
      asm p:$220 rnd a<return>
      change pc $220<return>
      step<return>
```

Results

```
sr=$00d0
a1=$111112    a0=$000000
```

EXAMPLE 3 - 1 3**Value of B-accumulator register B0 is less than 0.5***Problem*

Execute the instruction **RND B** to show that the B-Accumulator Register B1 is not changed when the value of the B-Accumulator Register B0 is less than 0.5 (\$800000). Assume that the contents of the B1 and B0 accumulator registers before execution of the instruction are:

b1=\$111111 b0=\$777777

Procedure

Type: change b1 \$111111 b0 \$777777<return>
 asm p:\$221 rnd b<return>
 change pc \$221<return>
 step<return>

Results

b1=\$111111 b0=\$000000

EXAMPLE 3 - 1 4**The Values of A- and B-accumulator registers A0 and B0 are both 0.5***Problem*

Execute the two instructions **RND A** and **RND B** to show that:

- (1) a one is added to the A-A ccumulator Register A1 when the least significant bit of A1 is equal to one and the value of A0 is equal to 0.5 (\$800000), and
- (2) B-Accumulator Register B1 is unchanged when the least significant bit of B1 is equal to zero and the value of B0 is equal to 0.5 (\$800000)

Assume that the contents of the A1, A0, B1, and B0 accumulator registers before execution of the two instructions are:

a1=\$111111 a0=\$800000
 b1=\$101010 b0=\$800000

Hit the function key F2 to change the default radix in the command window to decimal.

Procedure

Type: change a1 \$111111 a0 \$800000<return>
 change b1 \$101010 b0 \$800000<return>
 asm p:\$222 rnd a<return>
 asm p:\$223 rnd b<return>
 change pc \$222<return>
 step 2<return>

Results

a1=\$111112 a0=\$000000
 b1=\$101010 b0=\$000000

E X A M P L E 3 – 1 5

Multiplication with and without rounding*Problem*

Execute the instructions **MPYR X1,X0,A** and **MPY X1,X0,B** to:

- (1) multiply the value of the X1 Input Data Register by the value of the X0 Input Data Register and store the convergently rounded result in the A-Accumulator, and
- (2) multiply the value of the X1 Input Data Register by the value of the X0 Input Data Registers and store the nonrounded result in the B-Accumulator

Assume that the contents of the X1 and X0 data registers and the A and B accumulator before execution of the two instructions are:

```
x1 = $400000    x0 = $000007
a = $00000000000000
b = $00000000000000
```

Procedure

```
Type: change a 0 b 0<return>
      change x1 $400000 x0 $000007<return>
      asm p:$224 mpyr x1,x0,a<return>
      asm p:$225 mpy x1,x0,b<return>
      change pc $224<return>
      step 2 <return>
```

Results

```
x1=$400000    x0=$000007
a=$00000004000000
b=$00000003800000
```

Note that the contents of the A-Accumulator are convergently rounded and that the contents of the B-Accumulator are not.

3.3.3 Accumulator Shifter

The Accumulator Shifter accepts a 56-bit input and produces a 56-bit output. The Accumulator Shifter either shifts the input data operand one bit position to the left, one bit position to the right, or passes it unshifted.

E X A M P L E 3 – 1 6

Left shifting*Problem*

Execute the instruction **ASL A** to arithmetically shift the A-Accumulator one bit position to the left. Assume that the contents of the A-Accumulator and the Status Register (SR) before execution of the instruction are:

```
a=$ff888888333333=-0.933333373069765
sr=$00c0
```

Procedure

```
Type: change a $ff888888333333 sr $00c0<return>
asm p:$230 asl a<return>
change pc $230<return>
step<return>
```

Results

```
a=$ff111110666666=-1.866666746139529
sr=$00f9
```

Note that the Carry Bit of the Status Register (least significant bit) receives the most significant bit shifted out of the A-Accumulator and that a zero is shifted into the least significant bit of the A-Accumulator. Also note that the contents of the A-Accumulator are scaled up by a factor of two.

EXAMPLE 3 - 17

Right shifting

Problem

Execute the instruction **ASR B** to arithmetically shift the B-Accumulator one bit position to the right. Assume that the contents of the B-Accumulator and the Status Register (SR) before execution of the instruction are:

```
b=$ff888888333333=-0.933333373069765
sr=$00c0
```

Procedure

```
Type: change b $ff888888333333 sr $00c0<return>
asm p:$231 asr b<return>
change pc $231<return>
step<return>
```

Results

```
b=$ffc44444199999=-0.466666686534886
sr=$00d9
```

Note that the Carry Bit of the Status Register (least significant bit) receives the least significant bit shifted out of the B-Accumulator and that the most significant bit of the B-Accumulator is held constant. Also note that the contents of the B-Accumulator are scaled down by a factor of two.

3.3.4 Data Shifter/Limiters

Two Data Shifters/Limiters provide special post processing on data that is moved from the accumulators to the XDB or YDB. One 56-bit Shifter/Limiter is associ-

ated with the A-Accumulator; the other is associated with the B-Accumulator. Each shifter/limiter consists of a shifter followed by a limiter.

3.3.4.1 Data Shifters Each of two Data Shifters is capable of shifting a data operand one bit position to the left, one bit position to the right, or passing the data unshifted. One Data Shifter is associated with the A-Accumulator; the other is associated with the B-Accumulator. The output of each Data Shifter passes through an associated Data Limiter, where the Data Limiter has a 24-bit output and an overflow output indicator. The Data Shifters are controlled by the S1 and S0 Scaling Mode Bits in the Status Register (SR), where S1 and S0 are the eleventh and tenth bits, respectively. S1 and S0 permit dynamic scaling of fixed point data without the need for the user to modify the program's instructions. The scaling modes are shown in Table 3.1.

Table 3.1 Scaling Modes

S1	S0	Scaling Mode
0	0	No Scaling
0	1	Scale Down (1 bit arithmetic right shift)
1	0	Scale Up (1 bit arithmetic left shift)

EXAMPLE 3 – 18

No scaling

Problem

Execute the instructions **MPY Y1,X1,A** and **MOVE A,X0** to show that the data shifter is controlled by the Scaling Mode Bits (S1 and S0) in the Status Register. Assume that the contents of the X0, X1, and Y1 data registers, the A-Accumulator, and the Scaling Mode Bits before execution of the instruction are:

```
x1 = $400000
y1 = $400000
a = $0000000000000000
x0 = $000000
sr = $00c0
S1=0, S0=0 (no scaling)
```

Procedure

```
Type: change x1 $400000 y1 $400000<return>
      change sr $00c0 a 0 x0 0<return>
      asm p:$235 mpy y1,x1,a<return>
      asm p:$236 move a,x0<return>
      change pc $235<return>
      step 2<return>
```


Results

```

x1 = $400000
y1 = $400000
sr = $00d0
a  = $002000000000000
x0 = $200000

```

E X A M P L E 3 - 1 9**Down scaling***Problem*

Execute the instructions **MPY Y1,X1,A** and **MOVE A,X0** to show that the data shifter is controlled by the Scaling Mode Bits S1 and S0 in the Status Register. Assume that the contents of the X0, X1, and Y1 data registers, the A-Accumulator, and the Scaling Mode Bits before execution of the instruction are:

```

x1 = $400000
y1 = $400000
a  = $000000000000000
x0 = $000000
sr = $0400
S1=0, S0=1 (down scaling)

```

Procedure

```

Type: change a 0 pc $235 x0 0<return>
      change sr $0400<return>
      step 2<return>

```

Results

```

x1 = $400000
y1 = $400000
sr = $0410
a  = $002000000000000
x0 = $100000

```

Note that the contents of the X0 Data Input Register are down scaled by a factor of two.

E X A M P L E 3 - 2 0**Up scaling***Problem*

Execute the instructions **MPY Y1,X1,A** and **MOVE A,X0** to show that the data shifter is controlled by the Scaling Mode Bits S1 and S0 in the Status Register. Assume that the contents of the X0, X1, and Y1 data registers, the A-Accumulator, and the Scaling Mode Bits before execution of the instruction are:

```

x1 = $400000
y1 = $400000

```

```

a = $000000000000000
x0 = $000000
sr = $0800
S1=1, S0=0 (up scaling)

```

Procedure

```

Type: change a 0 pc $235 x0 0<return>
      change sr $0800<return>
      step 2<return>

```

Results

```

x1 = $400000
y1 = $400000
sr = $0880
a = $002000000000000
x0 = $400000

```

Note that the contents of the X0 Data Input Register are up scaled by a factor of two. The Scaling Bit in the Status Register (the seventh bit) is set to detect this data growth.

3.3.4.2 Limiters Each of two Data Limiters is capable of automatically performing, as necessary, saturation arithmetic on data operands that are moved from the accumulators to the XDB and YDB. If the contents of the selected source accumulator can be represented in the destination operand size without overflow, the data limiter is disabled and the operand is not modified. If the contents of the selected source accumulator cannot be represented without overflow in the destination operand size, the data limiter will substitute a *limited* data value having maximum magnitude and the same sign as the source accumulator. The limiting is performed on the output of the associated Data Shifter. The value of the source accumulator is not changed. One Data Limiter is associated with the A-Accumulator; the other Data Limiter is associated with the B-Accumulator.

With two Data Shifters/Limiters, two 24-bit operands can be limited independently in the same instruction cycle. Also, the two Data Shifters/Limiters can be concatenated to form one 48-bit limiter for long-word operands. Limited data values are shown in Table 3.2.

E X A M P L E 3 – 2 1

Transferring data with limiting*Problem*

Execute the two instructions **MOVE A,X0** and **MOVE B,Y1** to show that limiting occurs automatically when transferring data from the 56-bit A- and B-Accumulators to the 24-bit X0 and Y1 Data Input Registers, respectively. Assume that the contents of the A- and B-Accumulators and the X0 and Y1 Data Input Registers before execution of the two instructions are:

```

a=$0100000f800000
a2=$01 a1=$00000f a2=$800000

```

```

x0=$000000
b=$8000000f800004
b2=$80 b1=$00000f a2=$800004
y1=$000000

```

Procedure

```

Type: change a $0100000f800000 x0 $000000<return>
change b $8000000f800004 y1 $000000<return>
asm p:$237 move a,x0<return>
asm p:$238 move b,y1<return>
change pc $237<return>
step 2<return>

```

Results

```

a=$0100000f800000
a2=$01 a1=$00000f a2=$800000
x0=$7fffff
b=$8000000f800004
b2=$80 b1=$00000f a2=$800004
y1=$800000

```

Note that the contents of the X0 and Y1 registers are limited to \$7FFFFFFF and \$800000, respectively.

Table 3.2 Limited Data Values

Destination Memory Reference	Source Operand	Accumulator Sign	Limited Value (Hex)		Type of Access
			XDB	YDB	
X	X:A	+	7FFFFFFF	—	One 24-bit word
	X:B	-	800000	—	
Y	Y:A	+	—	7FFFFFFF	One 24-bit word
	Y:B	-	—	800000	
X and Y	X:A Y:A	+	7FFFFFFF	7FFFFFFF	Two 24-bit words
	X:A Y:B	-	800000	800000	
	X:B Y:A	+	7FFFFFFF	7FFFFFFF	
	X:B Y:B	-	800000	800000	
	L:AB	+	7FFFFFFF	7FFFFFFF	
	L:BA	-	800000	800000	
L (X:Y)	L:A	+	7FFFFFFF	FFFFFFF	One 48-bit long word
	L:B	-	800000	000000	

 E X A M P L E 3 – 2 2

Transferring data without limiting*Problem*

Execute the instruction **MOVE A1,X0** to show that limiting is not performed when the A1 (A2 or A0) Accumulator Register is specified as the source for a data move over XDB or YDB. Assume that the contents of the A-Accumulator and the X0 Data Input Register before execution of the instruction are:

```
a=$0080000000000000
a2=$00 a1=$800000 a2=$000000
x0=$000000
```

Procedure

```
Type: change a $0080000000000000 x0 $000000<return>
      asm p:$239 move a1,x0<return>
      change pc $239<return>
      step<return>
```

Results

```
a=$0080000000000000
a2=$00 a1=$800000 a2=$000000
x0=$800000
```

Note that the error between the contents of the A-Accumulator and the X0 Data Input Register after execution of the MOVE instruction is relatively large (1-(-1) = 2 decimal). To reduce this error, the contents of the full A-Accumulator, with limiting, should be transferred to the X0 Data Input Register. In that case the value of X0 would be \$7FFFFFFF and the error between the contents of the A-Accumulator and X0 would be relatively small (1 - 0.9999999 = 0.0000001 decimal).

3.3.5 Bit Manipulation Unit

The Bit Manipulation Unit performs bit manipulation operations on X-Memory or Y-Memory operands.

 E X A M P L E 3 – 2 3

Bit test and clear*Problem*

Execute the instruction **BCLR #2,X:\$100** to test and clear the second bit of X-Memory location \$100. Assume that the contents of X-Memory location \$100 and the Status Register (SR) before the instruction's execution are:

```
x:$100 $00000f sr=$00c0
```

Procedure

```
Type: change x:$100 $00000f sr $00c0<return>
```

```

display x:$100<return>
asm p:$240 bclr #2,x:$100<return>
change pc $240<return>
step<return>

```

Results

x:\$100 \$00000b sr=\$00c1

Note that the state of the second bit of X-Memory location \$100 is reflected in the Carry Bit of the Status Register (least significant bit). Also note that the second bit of X-Memory location \$100 is cleared after it is tested.

E X A M P L E 3 - 2 4**Bit test and change***Problem*

Execute the instruction **BCHG #3,Y:\$200** to test the third bit of Y-Memory location \$200. Assume that the contents of Y-Memory location \$200 and the Status Register (SR) before execution of the instruction are:

y:\$200 \$00000f sr=\$00c0

Procedure

```

Type: change y:$200 $00000f sr $00c0<return>
display y:$200<return>
asm p:$242 bchg #3,y:$200<return>
change pc $242<return>
step<return>

```

Results

y:\$200 \$000007 sr=\$00c1

Note that the state of the third bit of Y-Memory location \$200 is reflected in the Carry Bit of the Status Register (SR) and the state of the third bit of Y-Memory location \$200 is changed after it is tested.

3.4 PROGRAM CONTROLLER

The Program Controller, shown in Fig. 3.4, is an independent execution unit that provides standard program flow-control resources such as the Program Counter, Status Register, and System Stack. It also includes an Operating Mode Register, and a Loop Address Register and a Loop Counter dedicated to support of the DSP56002 processor's hardware DO loop instruction.

For more information on the Program Controller refer to the *DSP56000 Digital Signal Processing Family Manual*.

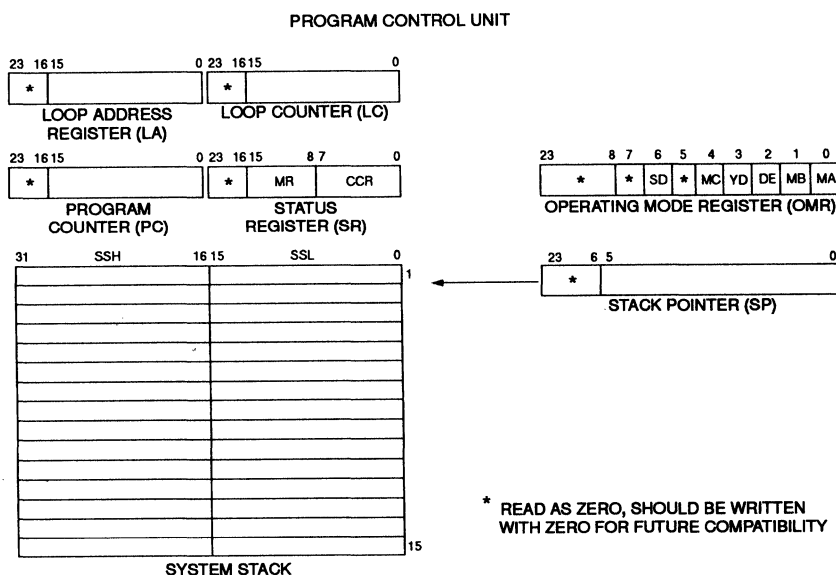


Fig. 3.4 Program Controller

Program Counter Register (PC): The 16-bit Program Counter Register (PC) points to the Program Memory (P-Memory) location of the next instruction word, immediate data operand, or immediate address operand.

Status Register (SR): The 16-bit Status Register (SR), shown in Fig. 3.5, consists of an 8-bit Mode Register (MR) and an 8-bit Condition Code Register (CCR). The MR is located in the high-order 8-bits of the Status Register; the CCR is located in the low-order 8-bits. The MR contains information about the system state of the DSP56002 processor; the CCR defines the current user state of the processor.

System Stack (SS): The System Stack (SS) is a separate 32 x 15 internal memory that stores the Program Counter (PC) and Status Register (SR) for subroutine calls, long interrupts, and program looping. The System Stack can also store the Loop Counter and Loop Address Register.

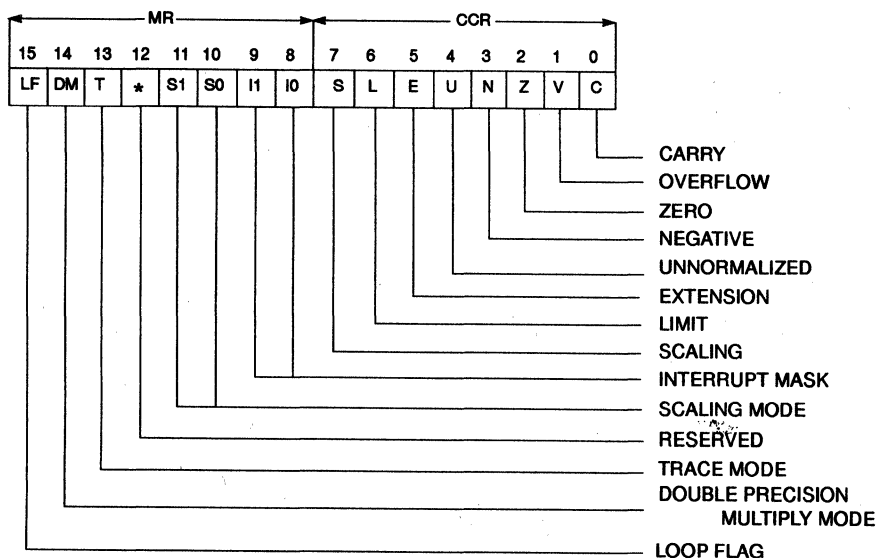
Stack Pointer (SP): The 6-bit Stack Pointer (SP), shown in Fig. 3.6, points to the location of the top of the System Stack and indicates the System Stack status conditions such as underflow, empty, full, and overflow.

Loop Counter (LC): The 16-bit Loop Counter (LC) specifies the number of times a hardware DO loop instruction or hardware REPEAT instruction is to be repeated.

Loop Address Register (LA): The 16-bit Loop Address (LA) Register points to the location of the last instruction word in a hardware DO loop.

Operating Mode Register (OMR): The 24-bit Operating Mode Register (OMR) defines the current operating mode of the DSP56002 processor. Only six bits of the OMR are defined. It defines how the various memories are mapped, and it defines the

startup procedure. The OMR format is shown in Fig. 3.7. Table 3.3 summarizes the DSP56002 processor's operating modes. Table 3.4 and Fig. 3.8 show the memory spaces.

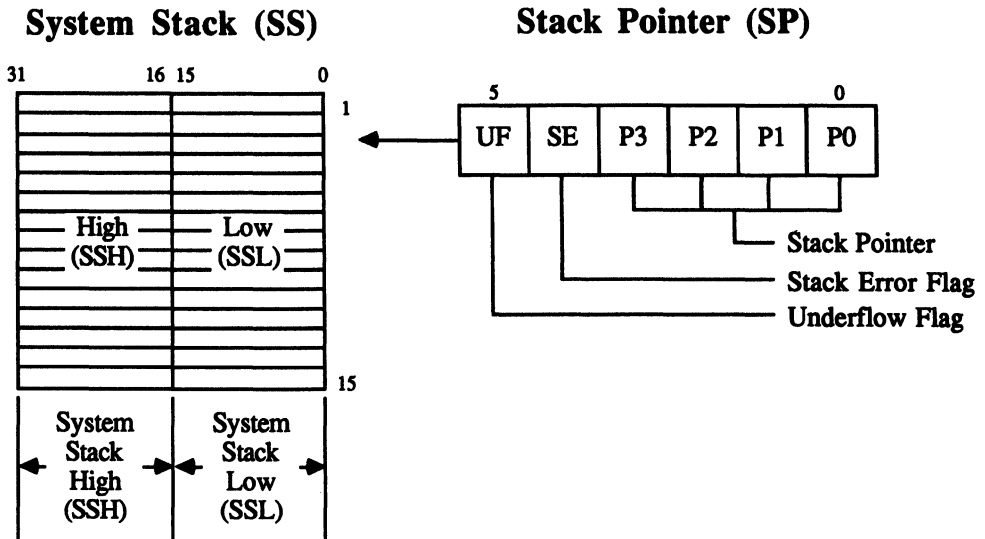


All bits are cleared after hardware reset except bits 8 and 9 which are set to ones.
Bits 12 and 16 to 23 are reserved, read as zero and should be written with zero for future compatibility

Fig. 3.5 Status Register

Table 3.3 DSP56002 Operating Mode Summary

Operating Mode	M C	M B	M A	Description
0	0	0	0	Single-Chip Mode—P:RAM enabled, reset at \$0000.
1	0	0	1	Bootstrap from EPROM, exit in Mode 0.
2	0	1	0	Normal Expanded Mode—P:RAM enabled, reset at \$E000.
3	0	1	1	Development Mode—P:RAM disabled, reset at \$0000.
4	1	0	0	Reserved for Bootstrap.
5	1	0	1	Bootstrap from Host, exit in Mode 0.
6	1	1	0	Bootstrap from SCI (external clock), exit in mode 0.
7	1	1	1	Reserved for Bootstrap.

**Uses:**

- Subroutines
- Long Interrupts
- Hardware "DO LOOPS"
- Not recommended for data storage

UF SE P3 P2 P1 P0

1	1	1	1	1	0	← Stack Underflow condition after double pull	} Stack Error Exception
1	1	1	1	1	1	← Stack Underflow condition	
0	0	0	0	0	0	← Stack Empty (reset). Pull causes underflow.	
0	0	0	0	0	1	← Stack location 1	
.	.	.	.	.	.		
.	.	.	.	.	.		
.	.	.	.	.	.		
0	0	1	1	1	0	← Stack location 14.	
0	0	1	1	1	1	← Stack location 15. Push causes overflow	
0	1	0	0	0	0	← Stack Overflow condition	} Stack Error Exception
0	1	0	0	0	1	← Stack Overflow condition after double push	

Fig. 3.6 Stack Pointer

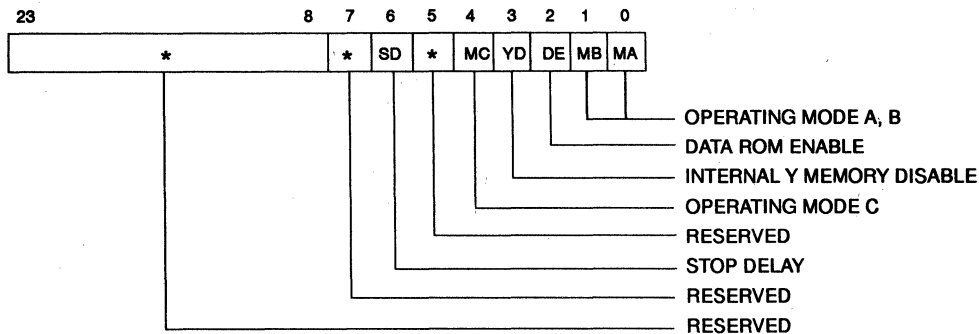


Fig. 3.7 Operating Mode Register

Table 3.4 Memory Mode Bits

XD	YD	Data Memory
0	0	Internal ROMs Disabled and their addresses are part of External Memory.
0	1	Internal X Data ROM is Disabled and is part of External Memory. Internal Y Data RAM and ROM are disabled and are part of External Memory.
1	0	X and Y Data ROMs Enabled.
1	1	Internal Y Data RAM and ROM are Disabled and are part of External Memory. Internal X Data ROM Enabled.

3.5 ADDRESS GENERATION UNIT

The Address Generation Unit is an independent execution unit that generates addresses to locate data operands in X-Memory, Y-Memory, or P-Memory. It provides fourteen addressing modes and uses three types of address generation arithmetic. Its major components, shown in Fig. 3.9, are: twenty-four 16-bit addressing registers, two address ALUs, and three address output multiplexers.

Address Registers: The 24 addressing registers, shown in Fig. 3.10, are organized into three sets of eight registers:

Address Registers	Rn n=0,1,...,7
Offset Registers	Nn n=0,1,...,7
Modifier Registers	Mn n=0,1,...,7

Each Address Register Rn has an associated Offset Register Nn and an associated Modifier Register Mn, the three having the same number n. The Address Registers Rn are used as address pointers to locate data operands in memory. The Offset Registers

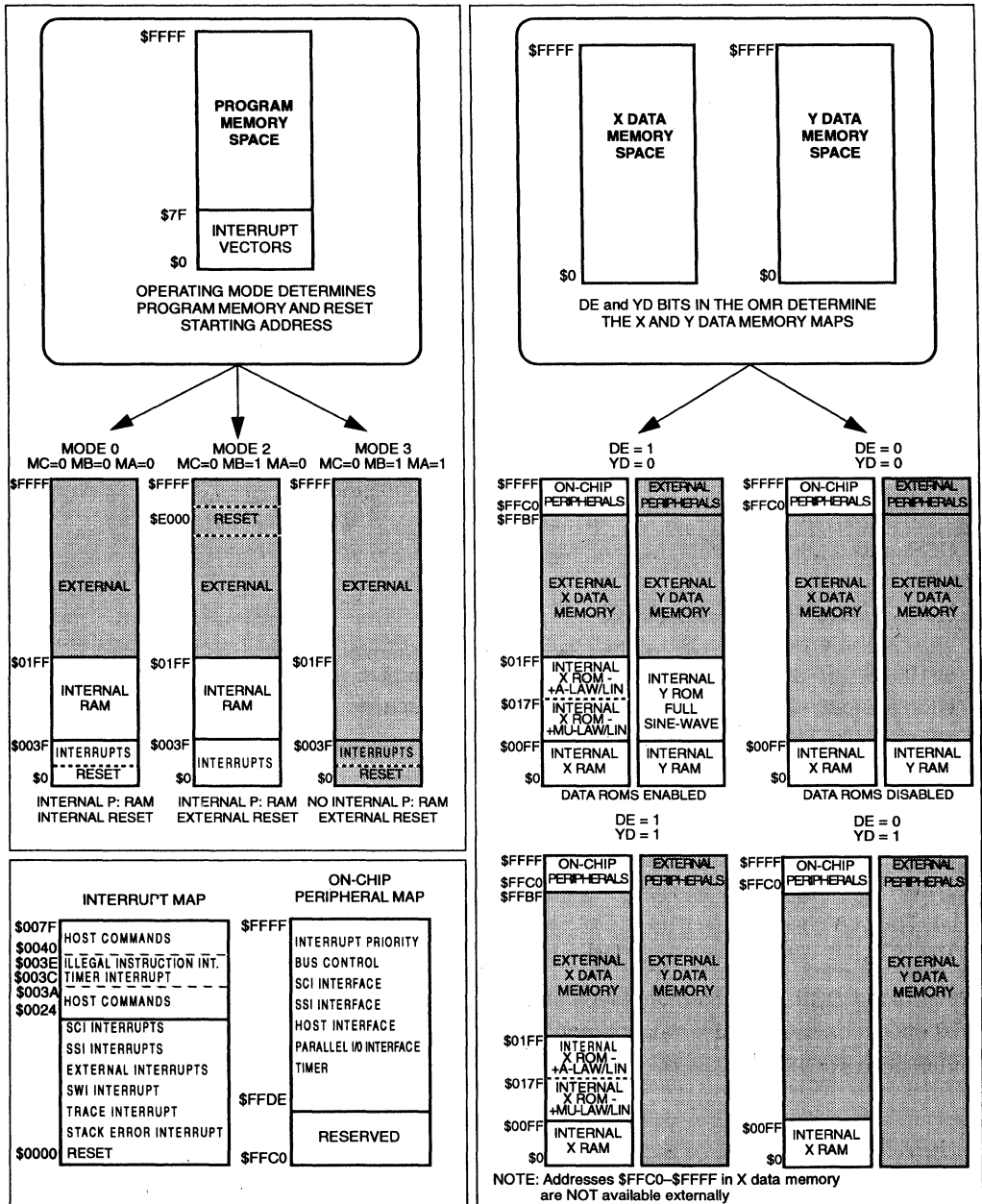


Fig. 3.8 DSP560002 Memory Maps

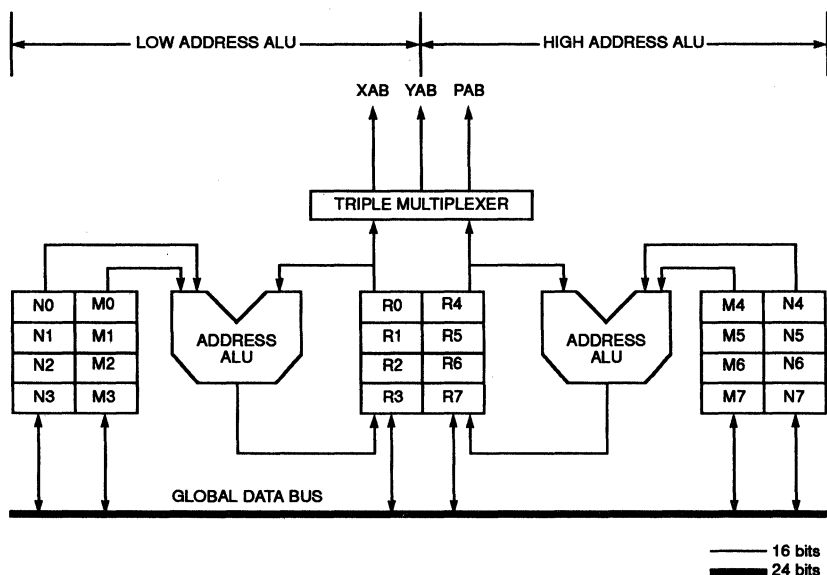
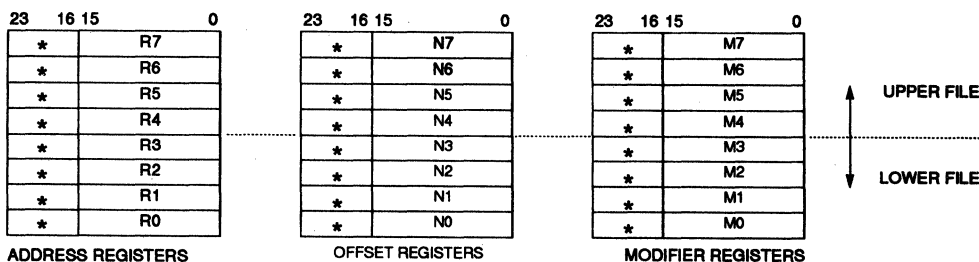


Fig. 3.9 Address Generation Unit



* Written as don't care; read as zero

Fig. 3.10 Address Registers

N_n are used to provide an offset value for offset updating of the Address Registers. The Modifier Registers M_n select the type of address arithmetic to be performed when an Address Register is updated.

Address ALUs: The two Address ALUs perform the address arithmetic by the DSP56002 processor's addressing modes and addressing modifiers. The Address Arithmetic Units use three types of address arithmetic: linear, modulo, and reverse-carry. The Modifier Registers define the type of address arithmetic to be performed. Linear address arithmetic is used for standard MPU-type addressing. Modulo address arithmetic is useful to support circular buffers as described in chapter 6. Reverse-carry arithmetic is useful for ordering data in Fast Fourier Transforms as described in

chapter 8. Table 3.5 shows how the contents of the Modifier Register select the type of address arithmetic to be performed.

Table 3.5 Modifier Type Encoding Summary

Modifier	Address Arithmetic Type
0000	Reverse Carry (Bit Reverse)
0001	Modulo 2
0002	Modulo 3
0003	Modulo 4
:	:
:	:
7FFE	Modulo 32767
7FFF	Modulo 32768
8000	Reserved
8001	Multiple Wrap-Around Modulo 2
8002	Reserved
8003	Multiple Wrap-Around Modulo 4
:	Reserved
8007	Multiple Wrap-Around Modulo 8
:	Reserved
800F	Multiple Wrap-Around Modulo 2 ⁴
:	Reserved
801F	Multiple Wrap-Around Modulo 2 ⁵
:	Reserved
803F	Multiple Wrap-Around Modulo 2 ⁶
:	Reserved
807F	Multiple Wrap-Around Modulo 2 ⁷
:	Reserved
80FF	Multiple Wrap-Around Modulo 2 ⁸
:	Reserved
81FF	Multiple Wrap-Around Modulo 2 ⁹
:	Reserved

Table 3.5 Modifier Type Encoding Summary (Continued)

Modifier	Address Arithmetic Type
83FF	Multiple Wrap-Around Modulo 2^{10}
:	Reserved
87FF	Multiple Wrap-Around Modulo 2^{11}
:	Reserved
8FFF	Multiple Wrap-Around Modulo 2^{12}
:	Reserved
9FFF	Multiple Wrap-Around Modulo 2^{13}
:	Reserved
BFFF	Multiple Wrap-Around Modulo 2^{14}
:	Reserved
FFFF	Linear (Modulo 2^{15})

Address Output Multiplexers: The three Address Output Multiplexers allow any Address Register Rn to be used as a pointer to any memory space.

3.6 ADDRESSING MODES

The DSP56002 processor's instructions consist of one or two 24-bit words: the operation word and the extension word. The operation word contains an 8-bit opcode field and a 16-bit data bus movement field. The opcode field includes the instruction opcode with its source and destination register operands. The data bus movement field provides the direction of transfer and the effective addresses for data movement on the XDB and YDB. The effective address specifies an addressing mode, and for some addressing modes the effective address will further specify an Address Register Rn.

The addressing modes specify whether the operands are in registers and/or memory locations and provide the specific address of the operands. The addressing modes can be grouped into three categories: Register Direct, Special, and Register Indirect. The Register Indirect modes require additional address modifier information that is specified in the Modifier Registers. Both the Register Direct and Register Indirect addressing modes are associated with single word instructions. The Special addressing modes are associated with one or two word instructions.

3.6.1 Register Direct Addressing Modes

The Register Direct addressing modes specify that the operand(s) is in one (or more) of the Data Input Registers, Address Registers, or Control Registers.

E X A M P L E 3 – 2 5

Problem

Execute the instruction **MOVE X1,A0** to move the contents of the X1 Data Input Register to the A0 Accumulator Register using the Register Direct addressing mode. Assume that the contents of the X1 Data Input Register and the A0 Accumulator Register before execution of the instruction are:

x1=\$111111 a0 = \$000000

Procedure

Type: change x1 \$111111 a0 0<return>
 asm p:\$100 move x1,a0<return>
 change pc \$100<return>
 step<return>

Results

x1=\$111111 a0 = \$111111

3.6.2 Special Addressing Modes

The Special address modes specify the operand or the address of the operand in a field of the instruction or they implicitly reference an operand. The Special addressing modes are: Immediate Data, Immediate Short Data, Absolute Address, Absolute Short Address, I/O Short Address, Short Jump Address, and Implicit. The Absolute Short, I/O Short, and Short Jump addresses, and the Immediate Short Data addressing modes are associated with single word instructions. The Absolute Address and Immediate Data addressing modes are associated with two word instructions.

Immediate Data Addressing Mode: The Immediate Data addressing mode points to a 24-bit operand located in the extension word of the instruction.

E X A M P L E 3 – 2 6

Transferring immediate data into a 24-bit accumulator register

Problem

Execute the instruction **MOVE #\$818181,A0** to transfer the immediate value \$818181 to Accumulator Register A0. Assume that the contents of the A-Accumulator before execution of the instruction are:

a= \$00000000000000
 a2=\$00 a1=\$000000 a0=\$000000

Procedure

Type: change a 0<return>
 asm p:\$101 move #\$818181,a0<return>

```
change pc $101<return>
step<return>
```

Results

```
a= $00000000818181
a2=$00 a1=$000000 a0=$818181
```

EXAMPLE 3 - 27

Transferring immediate data into the 56-bit accumulators

Problem

Execute the two instructions **MOVE #818181,A** and **MOVE #121212,B**. Assume that the contents of the A-Accumulator and B-Accumulator before execution of the two instructions are:

```
a= $0000000000000000
a2=$00 a1=$000000 a0=$000000
b= $0000000000000000
b2=$00 b1=$000000 b0=$000000
```

Procedure

```
Type: change a 0 b 0<return>
asm p:$103 move #818181,a<return>
asm p:$105 move #121212,b<return>
change pc $103<return>
step 2<return>
```

Results

```
a = $ff818181000000
a2=$ff a1=$818181 a0=$000000
b = $00121212000000
b2=$00 b1=$121212 b0=$000000
```

Note that Accumulator Registers A1 and B1 are loaded with the respective immediate data operands and that the Accumulator Registers A2 and B2 are sign-extended from A1 and B1, respectively.

Immediate Short Addressing Mode: The Immediate Short addressing mode points to an 8-bit or a 12-bit immediate data operand located in the instruction operation word. The immediate data is interpreted as an unsigned integer if the destination register is one of the Registers A2,A1,A0, B2,B1,B0, R0-R7, N0-N7. The immediate data is transferred to the least significant bits of the destination with the most significant bits zeroed. The immediate data is interpreted as a signed fraction if the destination is one of the Registers X1,X0, Y1,Y0, or the A- and B-Accumulators. The immediate data is transferred to the most significant bits of the destination with the least significant bits zeroed.

E X A M P L E 3 – 2 8

Transferring immediate short data into 24-bit registers

Problem

Execute the two instructions **MOVE #81,A1** and **MOVE #81,X1**. Assume that the contents of the Data Input Registers X1,X0 and the A-Accumulator before execution of the two instructions are:

```
a= $0000000000000000
a2=$00 a1=$000000 a0=$000000
x1=$000000 x0=$000000
```

Procedure

```
Type: change a 0 x 0<return>
asm p:$107 move #81,a1<return>
asm p:$108 move #81,x1<return>
change pc $107<return>
step 2<return>
```

Results

```
a = $00000081000000
a2=$00 a1=$000081 a0=$000000
x1=$810000 x0=$000000
```

Note that when Accumulator Register A1 is the destination, the immediate data 81 is interpreted as an unsigned integer and is transferred into the least significant bits of A1. When Data Input Register X1 is the destination, the immediate data 81 is interpreted as signed fraction and is transferred into the most significant bits of X1.

E X A M P L E 3 – 2 9

Transferring immediate short data into the accumulators

Problem

Execute the two instructions **MOVE #12,A** and **MOVE #81,B**. Assume that the contents of the A- and B-Accumulators before execution of the two instructions are:

```
a= $0000000000000000
a2=$00 a1=$000000 a0=$000000

b= $0000000000000000
b2=$00 b1=$000000 b0=$000000
```

Procedure

```
Type: change a 0 b 0<return>
asm p:$109 move #12,a<return>
asm p:$10a move #81,b<return>
change pc $109<return>
step 2<return>
```


Results

```

a = $0012000000000000
a2=$00 a1=$120000 a0=$000000

b = $ff81000000000000
b2=$ff b1=$810000 b0=$000000

```

Note that the immediate data operands \$12 and \$81 are interpreted as signed fractions and transferred into the most significant bits of the Accumulator Registers A1 and B1, respectively. Also note that Accumulator Registers A2 and B2 are sign-extended from A1 and B1, respectively.

Absolute Address Addressing Mode: The Absolute Address addressing mode uses the 16-bit address operand located in the instruction extension word as a pointer to the location of the data operand.

E X A M P L E 3 – 3 0

Transferring a data operand pointed to by an absolute address operand into the A-Accumulator
Problem

Execute the instruction **MOVE X:\$2000,A0** to transfer the contents of the X-Memory location pointed to by the instruction extension word to Accumulator Register A0. Assume that the contents of the A-Accumulator and X-Memory location \$2000 before execution of the instruction are:

```

a= $0000000000000000
a2=$00 a1=$000000 a0=$000000
x:$2000 $121212

```

Procedure

```

Type: change x:$2000 $121212 a 0<return>
      display x:$2000<return>
      asm p:$10b move x:$2000,a0<return>
      change pc $10b<return>
      step<return>

```

Results

```

a= $00000000121212
a2=$00 a1=$000000 a0=$121212
x:$2000 $121212

```

Absolute Short Addressing Mode: The Absolute Short addressing mode uses an immediate 6-bit address operand, located in the instruction operation word, to form a 16-bit pointer to the data operand. The 6-bit immediate address operand is zero-extended to form the 16-bit pointer.

E X A M P L E 3 – 3 1

Transferring an accumulator register to a memory location pointed to by an absolute short address operand
Problem

Execute the instruction **MOVE A1,X:\$2** to transfer the contents of Accumulator Register A1 to X-Memory location \$0002 pointed to by the 6-bit (zero-extended to 16-bits) Absolute Short address \$02. Assume that the contents of the A-Accumulator and X-Memory location \$0002 before execution of the instruction are:

```
a= $00818181000000
a2=$00 a1=$818181 a0=$000000
x:$0002 $000000
```

Procedure

```
Type: change x:$2 0 a 0 a1 $818181<return>
      display x:$2<return>
      asm p:$10d move a1,x:$2<return>
      change pc $10d<return>
      step<return>
```

Results

```
a = $00818181000000
a2=$00 a1=$818181 a0=$000000
x:$0002 $818181
```

I/O Short Addressing Mode: The I/O Short addressing mode is similar to the Absolute Short addressing mode. The I/O Short addressing mode uses an immediate 6-bit address operand, located in the instruction operation word, to form a 16-bit pointer to the data operand. The 6-bit immediate address operand is ones-extended to form a 16-bit pointer to the I/O sections (\$FFC0-\$FFFF) of X- or Y-Memory, respectively. It is used with the **bit manipulation** and **move peripheral data** instructions.

E X A M P L E 3 – 3 2

Transferring an accumulator register to an I/O memory location pointed to by an I/O short address operand
Problem

Execute the instruction **MOVEP A1,X:\$FFFE** to transfer the contents of Accumulator Register A1 to X-Memory I/O location \$FFFE pointed to by the 6-bit (ones-extended to 16-bits) I/O Short address \$3E. Assume that the contents of the A-Accumulator and X-Memory I/O location \$FFFE before execution of the instruction are:

```
a= $00112233000000
a2=$00 a1=$112233 a0=$000000
x:$ffe $000000
```

Procedure

```
Type: change x:$fffe 0 a 0 a1 $112233<return>
      display x:$fffe<return>
      asm p:$10e movep a1,x:$fffe<return>
      change pc $10e<return>
      step<return>
```

Results

```
a = $001122330000000
a2=$00 a1=$112233 a0=$000000
x:$fffe $002233
```

Note that the least significant 16-bits of Accumulator Register A1 is transferred to the X-Memory I/O location \$FFFE.

Short Jump Addressing Mode: The Short Jump addressing mode uses a 12-bit immediate jump operand, located in the instruction operation word, to form a 16-bit “jump to” operand. The 12-bit jump operand is zero-extended to 16-bits and replaces the contents of the Program Counter (PC).

E X A M P L E 3 – 3 3

Short jumping to a P-Memory location

Problem

Execute the instruction **JMP \$222** to jump to P-Memory location \$0222. Assume that the contents of the Program Counter before execution of the instruction are:

```
pc = $0200
```

Procedure

```
Type: change pc $200<return>
      asm p:$10f jmp $222<return>
      change pc $10f<return>
      step<return>
```

Result

```
pc = $0222
```

Implicit Addressing Mode: The Implicit addressing mode is used by some instructions to implicitly reference the Program Controller Registers. The Program Controller Registers are implied in the mnemonic and opcode of the instructions.

3.6.3 Address Register Indirect Addressing Modes

In the Address Register Indirect addressing modes, the instruction operation word specifies an Address Register Rn to point to an operand located in memory. The

instruction operation word can also specify an address calculation to be performed either pre- or postinstruction execution; e.g., postincrement by one, postdecrement by an offset.

Each Address Register Rn is associated with an Offset Register Nn and a Modifier Register Mn. The Offset Register Nn contains an offset value that can be added to Rn to update its contents. The Modifier Register Mn specifies the type of address arithmetic to be performed when Address Register Rn is updated. Each Modifier Register Mn is set to \$FFFF upon processor reset to specify linear address arithmetic as the default address arithmetic.

No Update Address Register Indirect Addressing Mode: In this Address Register Indirect addressing mode, Address Register Rn points to the operand located in memory. The contents of Address Register Rn are not changed.

EXAMPLE 3 – 3 4

Transferring an accumulator register to a memory location pointed to by an address register. The contents of the address register are not changed.

Problem

Execute the instruction **MOVE B1,Y:(R0)** to transfer the contents of Accumulator Register B1 to the Y-Memory location pointed to by Address Register R0. Assume that the contents of Accumulator Register B1, Address Register R0, Offset Register N0, Modifier Register M0, and Y-Memory location \$1200 before execution of the instruction are:

```
b1=$010101
r0=$1200 n0=$0000 m0=$ffff
y:$1200 $000000
```

Procedure

```
Type: change b1 $010101 y:$1200 0<return>
      change r0 $1200 n0 0 m0 $ffff<return>
      display y:$1200<return>
      asm p:$110 move b1,y:(r0)<return>
      change pc $110<return>
      step<return>
```

Results

```
b1=$010101
r0=$1200 n0=$0000 m0=$ffff
y:$1200 $010101
```

Post-increment by One Address Register Indirect Addressing Mode: In this Address Register Indirect addressing mode, Address Register Rn points to the operand located in memory. After the operand address is used, the contents of Address Register Rn are incremented by one and the result is stored in Rn.

E X A M P L E 3 - 3 5

Transferring an accumulator register to a memory location pointed to by an address register. The contents of the address register are postincremented by one.

Problem

Execute the instruction **MOVE B0,Y:(R1)+** to transfer the contents of Accumulator Register B0 to the Y-Memory location pointed to by Address Register R1. After the transfer is complete, R1 is incremented by one. Assume that the contents of Accumulator Register B0, Address Register R1, Offset Register N1, Modifier Register M1, and Y-Memory location \$1000 before execution of the instruction are:

```
b0=$010101
r1=$1000 n1=$0000 m1=$ffff
y:$1000 $000000
```

Procedure

```
Type: change b0 $010101 y:$1000 0<return>
      change r1 $1000 n1 0 m1 $ffff<return>
      display y:$1000<return>
      asm p:$111 move b0,y:(r1)+<return>
      change pc $111<return>
      step<return>
```

Results

```
b0=$010101
r1=$1001 n1=$0000 m1=$ffff
y:$1000 $010101
```

Postdecrement by One Address Register Indirect Addressing Mode: In this Address Register Indirect addressing mode, Address Register Rn points to the operand located in memory. After the operand address is used, the contents of Address Register Rn are decremented by one and the result is stored in Rn.

E X A M P L E 3 - 3 6

Transferring a data input register to a memory location pointed to by an address register. The contents of the address register are postdecremented by one.

Problem

Execute the instruction **MOVE Y0,X:(R2)-** to transfer the contents of Data Input Register Y0 to the X-Memory location pointed to by Address Register R2. After the transfer is complete, R2 is decremented by one. Assume that the contents of Data Input Register Y0, Address Register R2, Offset Register N2, Modifier Register M2, and X-Memory location \$1001 before execution of the instruction are:

```
y0=$010101
r2=$1001 n2=$0000 m2=$ffff
x:$1001 $000000
```

Procedure

```
Type: change y0 $010101 x:$1001 0<return>
      change r2 $1001 n2 0 m2 $ffff<return>
      display x:$1001<return>
      asm p:$112 move y0,x:(r2)-<return>
      change pc $112<return>
      step<return>
```

Results

```
y0=$010101
r2=$1000 n2=$0000 m2=$ffff
x:$1001 $010101
```

Postincrement by Offset Nn Address Register Indirect Addressing Mode: In this Address Register Indirect addressing mode, Address Register Rn points to the operand and located in memory. After the operand address is used, Rn is updated by adding the offset contained in Offset Register Nn to the contents of Rn. The contents of Offset Register Nn are unchanged.

E X A M P L E 3 – 3 7

Transferring a data input register to a memory location pointed to by an address register. The contents of the address register are postupdated by adding the offset value in the associated offset register.

Problem

Execute the instruction **MOVE X0,Y:(R3)+N3** to transfer the contents of Data Input Register X0 to the Y-Memory location pointed to by Address Register R3. After the transfer is complete, R3 is updated by adding the offset contained in Offset Register Nn to the contents of Rn. Assume that the contents of Data Input Register X0, Address Register R3, Offset Register N3, Modifier Register M3, and Y-Memory locations \$1000 and \$1003 before execution of the instruction are:

```
x0=$010101
r3=$1000 n3=$0003 m3=$ffff
y:$1000 $00000
y:$1003 $00000
```

Procedure

```
Type: change x0 $010101 <return>
      change y:$1000 0 y:$1003 0<return>
      change r3 $1000 n3 3 m3 $ffff<return>
      display y:$1000<return>
      asm p:$113 move x0,y:(r3)+n3<return>
      change pc $113<return>
      step<return>
```

Results

```

x0=$010101
r3=$1003 n3=$0003 m3=$ffff
y:$1000 $010101
y:$1003 $000000

```

Postdecrement by Offset Nn Address Register Indirect Addressing Mode: In this Address Register Indirect addressing mode, Address Register Rn points to the operand and located in memory. After the operand address is used, Rn is updated by subtracting the offset contained in Offset Register Nn from the contents of Rn. The contents of Offset Register Nn are unchanged.

E X A M P L E 3 – 3 8

Transferring a memory location pointed to by an address register to an accumulator register. The contents of the address register are postupdated by subtracting the offset value in the associated offset register.

Problem

Execute the instruction **MOVE Y:(R4)-N4,A0** to transfer the contents of the Y-Memory location pointed to by Address Register R4 to Accumulator Register A0. After the transfer is complete, R4 is updated by subtracting the offset contained in Offset Register Nn from the contents of Rn. Assume that the contents of Accumulator Register A0, Address Register R4, Offset Register N4, Modifier Register M4, and Y-Memory locations \$1000 and \$1004 before execution of the instruction are:

```

a0 = $000000
r4 = $1004 n4 = $0004 m4 = $ffff
y:$1000 $222222
y:$1004 $111111

```

Procedure

```

Type: change a0 0 <return>
      change y:$1004 $111111 y:$1000 $222222<return>
      change r4 $1004 n4 4 m4 $ffff<return>
      display y:$1000<return>
      asm p:$114 move y:(r4)-n4,a0<return>
      change pc $114<return>
      step<return>

```

Results

```

a0 = $111111
r4 = $1000 n4 = $0004 m4 = $ffff
y:$1000 $222222
y:$1004 $111111

```

Indexed by Offset Nn Address Register Indirect Addressing Mode: In this Address Register Indirect addressing mode, Address Register Rn is added to Offset Register

R_n to form a pointer to an operand located in memory. The contents of R_n and N_n are unchanged.

EXAMPLE 3 – 3 9

Transferring a data input register to a memory location pointed to by the sum of an address register and its associated offset register. The contents of the address register and offset register are not changed.

Problem

Execute the instruction **MOVE X1,Y:(R5+N5)** to transfer the contents of Data Input Register X1 to the Y-Memory location pointed to by the summation of the contents of the Address Register R5 and the Offset Register N5. Assume that the contents of Data Input Register X1, Address Register R5, Offset Register N5, Modifier Register M5, and Y-Memory locations \$1300 and \$1306 before execution of the instruction are:

```
x1 = $010101
r5 = $1300  n5 = $0006  m5 = $ffff
y:$1300 $000000
y:$1306 $000000
```

Procedure

```
Type: change x1 $010101 y:$1300 0 y:$1306 0<return>
      change r5 $1300  n5 $0006  m5 $ffff<return>
      display y:$1300<return>
      asm p:$115 move x1,y:(r5+n5)<return>
      change pc $115<return>
      step<return>
```

Results

```
x1= $010101
r5 = $1300  n5 = $0006  m5 = $ffff
y:$1300 $000000
y:$1306 $010101
```

Predecrement by One Address Register Indirect Addressing Mode: In this Address Register Indirect addressing mode, Address Register R_n points to the operand located in memory, but R_n is decremented by one before the operand is accessed; e.g., $R_n = R_n - 1$.

EXAMPLE 3 – 4 0

Transferring a memory location pointed to by a predecremented address register to an accumulator register. The contents of the address register are changed.

Problem

Execute the instruction **MOVE X:-(R6),A1** to predecrement Address Register R6 by one and transfer the X-Memory location pointed to by the decremented Address Register R6 to Accumulator Register A1. Assume that the contents of the A-Accumulator, Address Regis-

ter R6, Offset Register N6, Modifier Register N6, and X-Memory locations \$1000 and \$1001 before execution of the instruction are:

```
a=$0000000000000000
a2=$00 a1=$000000 a0=$000000
r6=$1001 n6=$0000 m6=$ffff
x:$1000 $121212 $343434
```

Procedure

```
Type: change a 0 x:$1000 $121212 x:$1001 $343434<return>
      change r6 $1001 n6 0 m6 $ffff<return>
      display x:$1000<return>
      asm p:$116 move x:-(r6),a1<return>
      change pc $116<return>
      step<return>
```

Results

```
a=$0012121200000000
a2=$00 a1=$121212 a0=$000000
r6=$1000 n6=$0000 m6=$ffff
x:$1000 $121212 $343434
```

3.6.4 DSP56002 Addressing Mode Summary

The DSP56002 processor's addressing modes are summarized in Table 3.6.

3.7 REFERENCE

1. Motorola Inc., *DSP56000 Digital Signal Processing Family Manual*, 1992.

Table 3.6 Summary of DSP56002 Addressing Modes

Addressing Mode	Assembler Syntax	Example
Register Direct		move x1,a0
Special		
Immediate Data	#xxxxxx	move #\$818181,a0
Immediate Short Data	#xx	move #\$81,a1 move #\$81,X1 move #\$12,a move #\$81,b
Absolute Address	xxxx	move x:\$2000,a0
Absolute Short Address	aa	move a1,x:\$2
I/O Short Address	pp	movep a1,x:\$fffe
Short Jump Address	xxx	jmp \$222
Implicit		
Special Register		
Indirect		
No Update	(Rn)	move b1,y:(r0)
Postincrement by 1	(Rn)+	move b0,y:(r1)+
Postdecrement by 1	(Rn)-	move y0,x:(r2)-
Postincrement by Offset Nn	(Rn)+Nn	move x0,y:(r3)+n3
Postdecrement by Offset Nn	(Rn)-Nn	move y:(r4)-n4,a0
Indexed by Offset Nn	(Rn+Nn)	move x1,y:(r5+n5)
Predecrement by 1	-(Rn)	move x:-(r6),a1

DSP56002 Instruction Set

This chapter describes, in detail, the DSP56002 processor's instruction set that consists of 67 instructions. Each instruction is one or two words in length. Thirty of the instructions can specify one or two data transfers to be executed in parallel with execution of the instruction's opcode. These data transfers are called "parallel-move operations." The instruction set is divided into the following groups: Move, Arithmetic, Logical, Bit Manipulation, Loop, and Program Control. The instruction set features many digital signal processing oriented instructions such as CMPM, NORM, RND, MACR, SUBL, SUBR, ADDL, ADDR, DO, and REP. The chapter begins by describing the instruction format and the parallel-move operations. The instruction set groups are then presented.

4.1 INSTRUCTION FORMAT

Each DSP56002 instruction consists of one or two words. The first word specifies the instruction and its length. The second word can contain an absolute address or immediate data.

4.1.1 One-Word Instruction

All the DSP56002 processor's addressing modes, except the immediate data and absolute address addressing modes, are available in one-word instructions. The

assembly language source code for a typical one-word instruction can be organized into four fields: opcode field, operands field, X-Bus data transfer field, and Y-Bus data transfer field.

Opcode: The opcode field typically indicates the data ALU operation to be performed; it can also specify a move, address generation, or program control operation.

Operands Field: The operands field specifies the operands to be used by the opcode.

Data Transfer Fields: The X-Bus and Y-Bus data transfer fields specify optional data transfers over the X- and Y-Buses and the associated addressing modes.

E X A M P L E 4 – 1

The following instructions are examples of one-word instructions:

Table 4.1 One-Word Instructions

Opcode	Operands Transfer	X-Bus Data Transfer	Y-Bus Data
MOVE	X1,A		
MOVE		X0,X:(R2)-	
MOVE			X0,Y:(R3)+N3
RND	A	X:(R0)+,R0	Y:(R4)+,Y0
MAC	X0,Y0,B	X:(R0)+,R0	Y:(R4)+,Y0
ADD	X,B	X:(R6)+,X1	Y0,Y:(R7)-

4.1.2 Two-Word Instruction

The absolute address and immediate data addressing modes are available in two-word instructions to provide a full 16-bit absolute address or a full 24-bit immediate data value.

E X A M P L E 4 – 2

The following instructions are examples of two-word instructions:

```
MOVE #$123456,A0
MOVE X:$2000,A0
```

The 24-bit immediate data operand (\$123456) and the 16-bit address operand (\$2000) are encoded in the second word of the instructions, respectively.

4.2 PARALLEL-MOVE OPERATIONS

Thirty DSP56002 instructions can specify one or two parallel data move operations to be executed in one instruction cycle, in parallel, with the instruction's opcode. Data moves from register to register, register to memory, and memory to register are allowed. If the A-Accumulator or B-Accumulator is the source register, then shifting or limiting may occur with the data transfer. If the A-Accumulator or B-Accumulator is the destination register, then sign extension and least significant zero fill occurs automatically with the data transfer. When two data transfers are executed in one instruction, duplicate sources are permitted, but duplicate destinations are not permitted.

E X A M P L E 4 – 3

(a) An instruction with the same source transferring to several destinations is permitted:

SUBL B,A B,X:(R0) B,Y:(R4)

(b) An instruction with several sources transferring to the same destination is not permitted:

ADD B,A X1,B Y1,A (Not Permitted)

4.3 PARALLEL-MOVE TYPES

The parallel-move types are:

1. Immediate Short Data Move
2. Register to Register Move
3. Address Register Update
4. X- or Y-Memory Move
5. X- or Y-Memory and Register Move
6. L-Memory Move
7. XY-Memory Move

Load the Debug-EVM Software program by following steps 5–6 of section 1.4.

4.3.1 Immediate Short Data Move

This parallel-move operation transfers an 8-bit immediate short operand to a destination register. The 8-bit operand can be interpreted as an unsigned integer or as a signed fraction, depending on the destination register.

a. *Unsigned Integer Immediate Short Operand:* An immediate short operand is interpreted as an unsigned integer if the destination register is A2, A1, A0, B2, B1, B0, R0..R7, or N0..N7. The 8-bit data is stored in the least significant bits of the destination register and the high order bits of the destination register are automatically reset to zero.

E X A M P L E4 - 4

Problem

Execute the instruction **ADD B,A** in parallel with the immediate short data move **#81,B0**. Assume that the contents of the A and B accumulator registers before execution of the instruction are:

```
a = $00111111000000
a2 = $00 a1 = $111111 a0 = $000000
b = $00222222ff0000
b2 = $00 b1 = $222222 b0 = $ff0000
```

Procedure

```
Type: change a $00111111000000<return>
      change b $00222222ff0000<return>
      asm p:$300 add b,a #$81,b0<return>
      change pc $300<return>
      step<return>
```

Results

```
a = $00333333ff0000
a2 = $00 a1 = $333333 a0 = $ff0000
b = $00222222000081
b2 = $00 b1 = $222222 b0 = $000081
```

Note that the immediate short operand \$81 is interpreted as an unsigned integer and is moved into the lower portion of b0, with the high order bits of b0 reset to zero.

b. Signed Fraction Immediate Short Operand: An 8-bit immediate short operand is interpreted as a signed fraction if the destination register is X0, X1, Y0, Y1, A, or B. The 8-bit operand is stored in the most significant bits of the destination register with the low order bits of the destination register automatically reset to zero.

E X A M P L E4 - 5

Problem

Execute the instruction **ADD B,A** in parallel with the immediate short data move **#81,B**. Assume that the contents of the A and B accumulator registers before execution of the instruction are:

```
a = $00111111000000
a2 = $00 a1 = $111111 a0 = $000000
b = $00222222222222
b2 = $00 b1 = $222222 b0 = $222222
```

Procedure

```
Type: change a $00111111000000<return>
      change b $00222222222222<return>
```

```
asm p:$301 add b,a #81,b<return>
change pc $301<return>
step<return>
```

Results

```
a = $00333333222222
a2 = $00 a1 = $333333 a0 = $222222
b = $ff810000000000
b2 = $ff b1 = $810000 b0 = $000000
```

Note that the immediate short operand \$81 is interpreted as a signed fraction and is moved into the higher order bits of b1. Register b2 is sign-extended from b1; the lower order bits of b1 and all of the bits of b0 are reset to zero.

4.3.2 Register to Register Data Move

This parallel-move operation transfers the contents of a source register to a destination register.

EXAMPLE 4 – 6

Problem

Execute the instruction **ADD B,A** in parallel with the register to register data move operation **X1,B**. Assume that the contents of the A and B accumulator registers and the input register X1 before execution of the instruction are:

```
a = $00111111000000
a2 = $00 a1 = $111111 a0 = $000000
b = $00123456111111
b2 = $00 b1 = $123456 b0 = $111111
X1 = $900000
```

Procedure

```
Type: change a $00111111000000 b $00123456111111<return>
change x1 $900000<return>
asm p:$302 add b,a x1,b<return>
change pc $302<return>
step<return>
```

Results

```
a = $00234567111111
a2 = $00 a1 = $234567 a0 = $111111
b = $ff900000000000
b2 = $ff b1 = $900000 b0 = $000000
x1 = $900000
```

Note that the destination register b0 is automatically zeroed and the destination register b2 is sign-extended from the destination register b1.

EXAMPLE4 - 7

Problem

Execute the two instructions **ADD B,A** and **ADD X1,A** with the register to register data move operations **B,X1** and **B,R0**, respectively. Assume that the contents of the A and B accumulator registers, the X1 data register, and the R0 address register before execution of the two instructions are:

```
a = $00111111000000
a2 = $00 a1 = $111111 a0 = $000000
b = $00800001000000
b2 = $00 b1 = $800001 b0 = $000000
x1 = $000000
r0 = $0000
```

Procedure

```
Type: change a $00111111000000 b $00800001000000<return>
      change x1 0 r0 0<return>
      asm p:$303 add b,a b,x1<return>
      asm p:$304 add x1,a b,r0<return>
      change pc $303<return>
      step 2<return>
```

Results

```
a = $ 01111111000000
a2 = $01 a1 = $111111 a0 = $000000
b = $00800001000000
b2 = $00 b1 = $800001 b0 = $000000
x1 = $7fffff
r0 = $ffff
```

Note that limiting is performed on the output of the B-Accumulator. The limited values of the destination registers X1 and R0 are \$7FFFFFFF and \$FFFF, respectively.

4.3.3 Address Register Update

This parallel-move operation updates the address register Rn according to the addressing mode.

EXAMPLE4 - 8

Problem

Execute the instruction **ADD B,A** in parallel with the address register update parallel move operation **(R1)+N1**, where the postincrement by offset Nn indirect addressing mode is used to update the R1 address register. Assume that the contents of the A and B accumulator registers, the R1 address register, and the N1 offset register before execution of the instruction are:

```
a = $00111111000000
b = $00222222000000
```

```

r1 = $1000
n1 = $0006

```

Procedure

```

Type: change a $00111111000000 b $00222222000000<return>
      change r1 $1000 n1 $0006<return>
      asm p:$305 add b,a (r1)+n1<return>
      change pc $305<return>
      step<return>

```

Results

```

a = $00333333000000
b = $00222222000000
r1 = $1006
n1 = $0006

```

4.3.4 X- or Y-Memory Data Move

This parallel-move operation transfers a one-word operand to/from X- or Y-Memory. The X- or Y-Memory to register, or register to X- or Y-Memory, direction can be specified.

E X A M P L E 4 – 9**Register to X-Memory***Problem*

Execute the instruction **ADD A,B** in parallel with the X-Memory data move operation **A,X:\$1000**, where the absolute address addressing mode is used to specify the X-Memory address \$1000. Assume that the contents of the A and B accumulator registers and the X-Memory location \$1000 before execution of the instruction are:

```

a = $00123456000000
a2 = $00 a1 = $123456 a0 = $000000
b = $00111111000000
b2 = $00 b1 = $111111 b0 = $000000
x:$1000 $000000

```

Procedure

```

Type: change a $00123456000000 b $00111111000000<return>
      change x:$1000 0<return>
      display x:$1000<return>
      asm p:$306 ADD A,B A,X:$1000<return>
      change pc $306<return>
      step<return>

```

Results

```

a = $00123456000000
a2 = $00 a1 = $123456 a0 = $000000
b = $00234567000000
b2 = $00 b1 = $234567 b0 = $000000
x:$1000 $123456

```

E X A M P L E 4 - 1 0

Y-Memory to register*Problem*

Execute the instruction **ADD A,B** in parallel with the Y-Memory data move operation **Y:-(R3),A**, where the predecrement by one addressing mode is used to specify the Y-Memory address. Assume that the contents of the A and B accumulator registers, the R3 address register, and the Y-Memory location \$1000 before execution of the instruction are:

```
a = $00123456000000
a2 = $00 a1 = $123456 a0 = $000000
b = $00111111000000
b2 = $00 b1 = $111111 b0 = $000000
y:$1000 $999999
r3 = $1001
```

Procedure

```
Type: change a $00123456000000 b $00111111000000<return>
change r3 $1001 y:$1000 $999999<return>
display y:$1000<return>
asm p:$308 ADD A,B y:-(r3),a<return>
change pc $308<return>
step<return>
```

Results

```
a = $ff999999000000
a2 = $ff a1 = $999999 a0 = $000000
b = $00234567000000
b2 = $00 b1 = $234567 b0 = $000000
y:$1000 $999999
r3 = $1000
```

4.3.5 X- or Y-Memory and Register Data Move

This parallel-move operation transfers a one-word operand to/from X- or Y-Memory and a one-word operand from register to register.

E X A M P L E 4 - 1 1

Problem

Execute the instruction **ADD X,A** in parallel with the X-Memory data move operation **A,X:(R3+N3)** and the register data move operation **A,Y1**. Then, execute the signed multiply instruction **MPY Y1,X1,A**. Assume that the contents of the A-Accumulator, the X and Y1 data registers, the R3 address register, the N3 offset register, and X-Memory location \$1006 before execution of the two instructions are:

```
x = $111111111111
```

```

a = $00123456000000
y1 = $000000
r3 = $1000 n3 = $0006
x:$1006 $000000

```

Procedure

The register-to-register move operation **A,Y1** allows the contents of the A-Accumulator register to be moved to the 24-bit Y1 data input register. The contents of Y1 will be used as an input operand in the **MPY Y1,X1,A** instruction. The multiplier input operands are each 24-bits wide.

```

Type: change x $111111111111 a $00123456000000<return>
      change r3 $1000 n3 $6 x:$1006 0 y1 0<return>
      display x:$1006<return>
      asm p:$309 add x,a a,x:(r3+n3) a,y1<return>
      asm p:$30a mpy y1,x1,a<return>
      change pc $309<return>
      step 2<return>

```

Results

```

x = $111111111111
a = $00026d60ca5f6c
y1 = $123456
r3 = $1000 n3 = $0006
x:$1006 $123456

```

4.3.6 Long Memory Data Move

This parallel-move operation transfers one long-word operand to/from L (X:Y) memory. Two Data ALU registers are concatenated to form the long-word operand. This allows one double precision (high:low) data value or one complex (real:imaginary) data value to be transferred to/from one L-Memory location.

E X A M P L E 4 - 1 2

Problem

Execute the instruction **ADD Y,B** in parallel with the long memory move operation **B10,L:\$1000**, where the absolute address addressing mode is used to specify the L-Memory address \$1000. Assume that the contents of the B-Accumulator register, the Y data register, and the L-, X-, and Y-memory locations \$1000 before execution of the instruction are:

```

y = $111111222222
b = $001111111111
b2= $00 b1= $111111 b0= $111111
l:$1000 $000000000000
x:$1000 $000000
y:$1000 $000000

```

Procedure

```
Type: change y $111111222222 b $0011111111111111<return>
      change l:$1000 $00000000000000<return>
      display l:$1000<return>
      asm p:$30b add y,b b10,l:$1000<return>
      change pc $30b<return>
      step<return>
```

Results

```
y = $111111222222
b = $00222222333333
l:$1000 $111111111111
x:$1000 $111111
y:$1000 $111111
```

Note that the 48-bit value in the b1 and b0 accumulator registers is stored as a double precision value in the L-Memory location \$1000.

4.3.7 XY-Memory Data Move

This parallel-move operation transfers two single-word operands to/from X- and Y-Memory.

E X A M P L E 4 - 1 3*Problem*

Execute the instruction **ADD X,B** in parallel with the XY memory move operations **X:(R0)+,X1** and **Y0,Y:(R7)-**, where the postincrement by one and postdecrement by one addressing modes are used to specify the X- and Y-Memory locations, respectively. Assume that the contents of the B-Accumulator, X and Y0 data registers, the R0 and R7 address registers, the X-Memory location \$1000, and the Y-Memory location \$1500 before execution of the instruction are:

```
x = $111111222222
y0 = $333333
b = $00111111000000
r0 = $1000 r7 = $1500
x:$1000 $444444 y:$1500 $000000
```

Procedure

- (1) Select a second data window (data window #2) by clicking the mouse on menu, view, data2, and done, respectively. Hit <alt-1> to select data window #1 and <alt-2> to select data window #2.
- (2) Type:

```
change x $111111222222 b $00111111000000<return>
change r0 $1000 r7 $1500 y0 $333333<return>
change x:$1000 $444444 y:$1500 0<return>
display x:$1000<return>
<alt-2> (remember to select the command window with the mouse click)
```

```

display y:$1500<return>
asm p:$30c add x,b x:(r0)+,x1 y0,y:(r7)-<return>
change pc $30c<return>
step<return>
<alt-1> (remember to select the command window with the mouse click)

```

Results

```

x = $444444222222
y0 = $333333
b = $002222222222
r0 = $1001  r7 = $14ff
x:$1000 $444444  y:$1500 $333333

```

4.3.8 Parallel-Moves Summary

Table 4.2 summarizes the parallel-move operations.

Table 4.2 Summary of Parallel-Move Operations

Parallel-Move Operations	Opcode Operands	Parallel-Move Examples
Immediate Short Data	ADD B,A	#\$81,B0
	ADD B,A	#\$81,B
Register to Register	ADD B,A	X1,B
	ADD B,A	B,X1
	ADD X1,A	B,R0
Address Register Update	ADD B,A	(R1)+N1
X or Y Memory	ADD A,B	A,X:\$1000
	ADD A,B	Y:-(R3),A
X or Y Memory plus register	ADD X,A	A,X:(R3+N3) A,Y1
L Memory	ADD Y,B	B10,L:\$1000
XY Memory	ADD X,B	X:(R0)+,X1 Y0,Y:(R7)-

The addressing modes that may be used with the parallel-move operations are summarized in Table 4.3, where:

U = Address Register Update
 X = X-Memory Move
 Y = Y-Memory Move
 X+R = X-Memory and Register Move

Y+R = Y-Memory and Register Move

L = L-Memory Move

XY = XY-Memory Move

Table 4.3 Addressing Modes for the U, X, Y, X+R, Y+R, L, and XY Parallel-Moves

Addressing Mode	Parallel-Moves				
	U	X Y	X+R Y+R	L	XY
Register Direct	No	No	No	No	No
Address Register Indirect					
No Update	No	Yes	Yes	Yes	Yes
Postincrement by 1	Yes	Yes	Yes	Yes	Yes
Postdecrement by 1	Yes	Yes	Yes	Yes	Yes
Postincrement by Offset Nn	Yes	Yes	Yes	Yes	Yes
Postdecrement by Offset Nn	Yes	Yes	Yes	Yes	No
Indexed by Offset Nn	No	Yes	Yes	Yes	No
Predecrement by 1	No	Yes	Yes	Yes	No
Special					
Immediate Data	No	Yes	Yes	No	No
Absolute Address	No	Yes	Yes	Yes	No
Immediate Short Data	No	No	No	No	No
Short Jump Address	No	No	No	No	No
Absolute Short Address	No	Yes	No	Yes	No
I/O Short Address	No	No	No	No	No
Implicit	No	No	No	No	No

4.4 DSP56002 INSTRUCTION SET

The DSP56002 instruction set is divided into the following groups:

1. Move

2. Arithmetic
3. Logical
4. Bit Manipulation
5. Loop
6. Program control

4.4.1 Move Instructions

The Move instructions transfer data over the XDB and YDB, the program data bus, and the Global Data Bus. The Move instructions can be considered to be a data ALU NOP (no operation) with the ability to perform parallel-move operations. The Move instruction affects the limit bit L and the scaling bit S in the Condition Code Register CCR, where the limit bit is the sixth bit and the scaling bit is the seventh bit. The CCR occupies the low-order 8-bits of the Status Register (SR). The L bit is affected if limiting is performed when reading a data ALU accumulator register. The S bit is affected if data growth is detected when the A or B accumulator registers are moved onto the bus. The Move instructions are:

LUA	Load updated address
MOVE	Move data
MOVEC or MOVE	Move control register
MOVEM or MOVE	Move program memory
MOVEP	Move peripheral data

E X A M P L E 4 – 1 4

Load updated address (LUA)

Problem

Execute the instruction **LUA (R0)+N0, N1** to update Address Register R0 and store the updated address in Offset Register N1. Assume that the contents of Address Register R0 and Offset Registers N0 and N1 before execution of the instruction are:

r0 = \$1000 n0 = \$0006 n1 = \$0000

Procedure

```
Type: change r0 $1000 n0 $6 n1 0<return>
      asm p:$400 lua (r0)+n0,n1<return>
      change pc $400<return>
      step<return>
```

Results

r0 = \$1000 n0 = \$0006 n1 = \$1006

E X A M P L E 4 - 1 5

Move control register*Problem*

Execute the instruction **MOVE A,LC (or MOVEC A,LC)** to transfer the A-Accumulator to the Loop Control (LC) Register. Assume that the contents of the Loop Control Register, the Status Register (SR), and the A-Accumulator before execution of the instruction are:

```
lc = $0000
sr = $0000
a = $00800001000000
```

Procedure

```
Type: change a 0 b 0<return>
      change lc $0 sr $0<return>
      change a $00800001000000<return>
      asm p:$401 move a,lc<return>
      change pc $401<return>
      step<return>
```

Results

```
lc = $ffff
sr = $0040
a = $00800001000000
```

Note that limiting is performed on the output of the A-Accumulator. The limited value transferred to the Loop Control Register is \$FFFF. The limit bit in the Status Register is set, where the limit bit is the sixth bit.

E X A M P L E 4 - 1 6

Move program memory*Problem*

Execute the two instructions **MOVE R3,P:(R2)-N2 (or MOVEM R3,P:(R2)-N2)** and **MOVE P:\$0000,LC (or MOVEM P:\$0000,LC)** to transfer the contents of Address Register R3 to the P-Memory location pointed to by Address Register R2; and transfer the contents of P-Memory location \$0000 to the Loop Counter (LC) Register. Assume that the contents of Address Registers R2 and R3, Offset Register N2, P-Memory locations \$0 and \$500, and the Loop Counter Register before execution of the two instructions are:

```
r3 = $1000 r2 = $0500 n2 = $0002
p:$0000 $123456
p:$0500 $000000
lc = $0000
```

Procedure

- (1) Select a second data window (data window #2) by clicking the mouse on menu, view, data2, and done, respectively. Hit <alt-1> to select data window #1 and <alt-2> to select data window #2.

```
(2) Type: change r3 $1000 r2 $500 n2 $2<return>
change p:$500 0 p:$0 $123456 lc 0<return>
display p:$500<return>
<alt-2> (remember to select the command window with the mouse click)
display p:$0<return>
asm p:$402 move r3,p:(r2)-n2<return>
asm p:$403 move p:$0000,lc<return>
change pc $402<return>
step 2<return>
<alt-1> (remember to select the command window with the mouse click)
```

Results

```
r3 = $1000 r2 = $04fe n2 = $0002
p:$0000 $123456
p:$0500 $001000
lc = $3456
```

E X A M P L E

4 - 1 7

Move peripheral data (MOVEP)*Problem*

Execute the instruction **MOVEP X:\$FFFE,A** to transfer the contents of X-Memory I/O peripheral location \$FFFE to the A-Accumulator. Assume that the contents of X-Memory I/O peripheral location \$FFFE and the A-Accumulator before execution of the instruction are:

```
a = $0000000000000000
a2 = $00 a1 = $000000 a2 = $000000
x:$fffe = $00ffff
```

Procedure

```
Type: change x:$fffe $ffff a 0<return>
display x:$fffe<return>
asm p:$404 movep x:$fffe,a<return>
change pc $404<return>
step<return>
```

Results

```
a = $0000ffff00000000
a2 = $00 a1 = $00ffff a2 = $000000
x:$fffe = $00ffff
```

4.4.2 Arithmetic Instructions

The Arithmetic instructions use the Data ALU to perform all arithmetic-type operations. Source operands for the Arithmetic instructions are contained in the Data ALU's Input Registers or Accumulators; destinations for the results generated by exe-

cution of the Arithmetic instructions are either the A-Accumulator or the B-Accumulator. The Arithmetic instructions allow up to two parallel-move operations to be executed concurrently with execution of the instruction's opcode. The parallel-move operations use the XDB and YDB. As a result, the parallel-move operations can transfer new prefetched data from X- and/or Y-Memory to the Data ALU for use in subsequent instructions, and can transfer the results generated by the execution of previous instructions from the Data ALU to X- and/or Y-Memory.

The Arithmetic instructions execute in one instruction cycle and can affect all of the bits in the Condition Code Register. The Arithmetic instructions are:

ABS	Absolute value
ADC	Add with carry
ADD	Add
ADDL	Shift left then add
ADDR	Shift right then add
ASL	Arithmetic shift left
ASR	Arithmetic shift right
CLR	Clear
CMP	Compare
CMPM	Compare magnitude
DEC	Decrement by one
DIV	Divide iteration
INC	Increment by one
MAC	Multiply/accumulate
MACR	Multiply/accumulate and round
MPY	Multiply
MPYR	Multiply and round
NEG	Negate
NORM	Normalize iteration
RND	Round
SBC	Subtract with carry
SUB	Subtract
SUBL	Shift left then subtract
SUBR	Shift right then subtract
Tcc	Transfer Conditionally
TFR	Transfer
TST	Test

In the following Arithmetic instruction examples, the default no-scaling mode is used. For example, the scaling mode bits S1 and S0 are cleared, where S1 and S0 are the fourth and third bits, respectively, in the Mode Register (MR) and where the MR occupies the upper 8-bit portion of the Status Register (SR). To clear S0 and S1, the Debug-EVM Software program's CHANGE command can be used to modify the contents of the MR register (Example 4.18), or the **AND immediate to control register** instruction **andi #f3,mr** can be executed to logic AND the contents of the MR with the 8-bit immediate operand \$f3, and store the result in the MR (Example 4.30).

a. *Addition and Subtraction Arithmetic Instructions*: The addition and subtraction Arithmetic instructions are: **add with carry (ADC)**, **add (ADD)**, **shift left then add (ADDL)**, **shift right then add (ADDR)**, **subtract with carry (SBC)**, **subtract (SUB)**, **shift left then subtract (SUBL)**, and **shift right then subtract (SUBR)**. The addition and subtraction arithmetic instructions are summarized in Table 4.4.

Table 4.4 Addition and Subtraction Arithmetic Instructions

Mnemonic	Operation	Assembler Syntax	Source S	Destination D
ADC	$S+D+C^* \rightarrow D$	ADC S,D [PM] [†]	X,Y	A,B
ADD	$S+D \rightarrow D$	ADD S,D [PM]	A	B
			B	A
			X,X1,X0	A,B
			Y,Y1,Y0	A,B
ADDL	$S+2D \rightarrow D$	ADDL S,D [PM]	A	B
			B	A
ADDR	$S+D/2 \rightarrow D$	ADDR S,D [PM]	A	B
			B	A
SBC	$D-S-C \rightarrow D$	SBC S,D [PM]	X,Y	A,B
SUB	$D-S \rightarrow D$	SUB S,D [PM]	A	B
			B	A
			X,X1,X0	A,B
			Y,Y1,Y0	A,B
SUBL	$2D-S \rightarrow D$	SUBL S,D [PM]	A	B
			B	A
SUBR	$D/2-S \rightarrow D$	SUBR S,D [PM]	A	B
			B	A

*C = Carry Bit

[†]PM = Parallel Move

E X A M P L E 4 - 1 8

Shift left then subtract*Problem*

Execute the instruction **SUBL B,A** to arithmetically shift the A-Accumulator one bit position to the left with a zero shifted into the least significant bit, subtract the B-Accumulator from the shifted A-Accumulator, and store the result ($2A - B$) in the A-Accumulator. Assume that the contents of the A-Accumulator and B-Accumulator before execution of the instruction are:

```
a = $00222222222222
a2 = $00 a1 = $222222 a2 = $222222
b = $00111111111111
b2 = $00 b1 = $111111 b2 = $111111
```

Procedure

```
Type: change a 0 b 0<return>
      change sr $0300<return>
      change b $00111111111111 a $00222222222222<return>
      asm p:$500 subl b,a<return>
      change pc $500<return>
      step<return>
```

Results

```
a = $00333333333333
a2 = $00 a1 = $333333 a2 = $333333
b = $00111111111111
b2 = $00 b1 = $111111 b2 = $111111
```

E X A M P L E 4 - 1 9

Add long with carry*Problem*

Execute the instruction **ADC Y,A** to add the 48-bit Y Data Input Register plus the Carry Bit C to the A-Accumulator, and store the result in the A-Accumulator. The Carry Bit C is the least significant bit in the Status Register (SR). Assume that the contents of the 48-bit Y Data Input Register, the A-Accumulator, and the Status Register before execution of the instruction are:

```
y = $222222000000
a = $00555555000000
sr = $0001
```

Procedure

```
Type: change a 0 b 0<return>
      change sr $0001<return>
      change y $222222000000 a $00555555000000<return>
```

```
asm P:$501 adc y,a<return>
change pc $501<return>
step<return>
```

Results

```
y = $222222000000
a = $00777777000001
sr = $0080
```

Note that a new Carry Bit, resulting from execution of the above instruction, is now in the SR. The carry bit is cleared after the execution. The Scaling Bit, the seventh bit in the SR, is set to detect the data growth.

EXAMPLE 4 – 20

Shift right then add

Problem

Execute the instruction **ADDR B,A** to arithmetically shift the A-Accumulator one bit position to the right with the most significant bit held constant, add the B-Accumulator to the shifted A-Accumulator, and store the result ($A/2 + B$) in the A-Accumulator. Assume that the contents of the A-Accumulator and B-Accumulator before execution of the instruction are:

```
a = $00555555555555
a2 = $00 a1 = $555555 a0 = $555555
b = $00222222222222
b2 = $00 b1 = $222222 b0 = $222222
```

Procedure

```
Type: change a $00555555555555 b $00222222222222<return>
asm p:$502 addr b,a<return>
change pc $502<return>
step<return>
```

Results

```
a = $004ccccccccc
a2 = $00 a1 = $4cccc a0 = $cccc
b = $00222222222222
b2 = $00 b1 = $222222 b0 = $222222
```

b. Multiplication Arithmetic Instructions: The multiplication Arithmetic instructions are: **multiply-accumulate (MAC)**, **multiply-accumulate and round (MACR)**, **multiply (MPY)**, and **multiply and round (MPYR)**. The multiplication Arithmetic instructions are summarized in Table 4.5. The destination (D) for the results generated by the execution of these instructions is either the A-Accumulator or the B-Accumulator.

Table 4.5 Multiplication Arithmetic Instructions

Mnemonic	Operation	Assembler Syntax	Sources	
			S1	S2
MAC	$D \pm (S1 * S2) \rightarrow D$	MAC $\pm S1, S2, D$ [PM]*	X0	X0
	$D \pm (S1 * 2^{-n}) \rightarrow D$	MAC $\pm S1, \#n, D$	Y0	Y0
			X1	X0
			Y1	Y0
			X0	Y1
			Y0	X0
			X1	Y0
			Y1	X1
MACR	$D \pm (S1 * S2) + r^{\dagger} \rightarrow D$	MACR $\pm S1, S2, D$ [PM]	X0	X0
	$D \pm (S1 * 2^{-n}) + r \rightarrow D$	MAC $\pm S1, \#n, D$	Y0	Y0
			X1	X0
			Y1	Y0
			X0	Y1
			Y0	X0
			X1	Y0
			Y1	X1
MPY	$\pm (S1 * S2) \rightarrow D$	MPY $\pm S1, S2, D$ [PM]	X0	X0
	$\pm (S1 * 2^{-n}) \rightarrow D$	MPY $\pm S1, \#n, D$	Y0	Y0
			X1	X0
			Y1	Y0
			X0	Y1
			Y0	X0
			X1	Y0
			Y1	X1
MPYR	$\pm (S1 * S2) + r \rightarrow D$	MPYR $\pm S1, S2, D$ [PM]	X0	X0
	$\pm (S1 * 2^{-n}) + r \rightarrow D$	MPYR $\pm S1, \#n, D$	Y0	Y0
			X1	X0
			Y1	Y0
			X0	Y1
			Y0	X0
			X1	Y0
			Y1	X1

*PM = Parallel Move

 $^{\dagger}r$ = Round

E X A M P L E 4 – 2 1

Signed multiply without and with rounding*Problem*

Execute the two instructions **MPY Y1,X1,A** and **MPYR Y1,X1,B** to multiply without and with rounding, respectively, the two signed operands Y1 and X1, and store the products in the A-Accumulator and B-Accumulator, respectively. Assume that the contents of the Y1 and X1 Data Input Registers, the A-Accumulator, and the B-Accumulator before execution of the two instructions are:

```

y1 = $400000
x1 = $000007
a = $00000000000000
a2 = $00 a1 = $000000 a0 = $000000
b = $00000000000000
b2 = $00 b1 = $000000 b0 = $000000

```

Procedure

```

Type: change x1 $000007 y1 $400000 a 0 b 0<return>
asm p:$503 mpy y1,x1,a<return>
asm p:$504 mpyr y1,x1,b<return>
change pc $503<return>
step 2<return>

```

Results

```

y1 = $400000
x1 = $000007
a = $00000003800000
a2 = $00 a1 = $000003 a0 = $800000
b = $00000000400000
b2 = $00 b1 = $000004 b0 = $000000

```

Note that the product in the B-Accumulator is a convergently rounded version of the product in the A-Accumulator.

E X A M P L E 4 – 2 2

Signed multiply-accumulate with and without rounding*Problem*

Execute the two instructions **MAC -Y1,X1,A** and **MACR -Y1,X1,B** to multiply without and with rounding, respectively, the two signed operands Y1 and X1; subtract the product from the A-Accumulator and B-Accumulator, respectively; and store the results in the A-Accumulator and the B-Accumulator, respectively. Assume that the contents of the Y1 and X1 Data Input Registers, the A-Accumulator, and the B-Accumulator before execution of the two instructions are:

```

y1 = $000007
x1 = $400000

```



```
a = $00000005100000
b = $00000005100000
```

Procedure

```
Type: change x1 $400000 y1 $000007<return>
      change a $00000005100000 b $00000005100000<return>
      asm p:$505 mac -y1,x1,a<return>
      asm p:$506 macr -y1,x1,b<return>
      change pc $505<return>
      step 2<return>
```

Results

```
y1 = $000007
x1 = $400000
a = $00000001900000
b = $00000002000000
```

The product generated by multiplying Y1 by X1 (\$00000003800000) is subtracted from the A- and B-Accumulators (\$00000005100000). Note that the result in the B-Accumulator (\$00000002000000) is a convergently rounded version of the result in the A-Accumulator (\$00000001900000).

c. *Other Arithmetic Instructions:* The **transfer data ALU register (TFR)**, **negate accumulator (NEG)**, **absolute value (ABS)**, **normalize accumulator iteration (NORM)**, and **clear accumulator (CLR)** instructions are summarized in Table 4.6.

EXAMPLE 4 - 2 3

Transfer data ALU register

Problem

Execute the instruction **TFR A,B** to transfer data from the A-Accumulator to the B-Accumulator. Assume that the contents of the A-Accumulator and the B-Accumulator before execution of the instruction are:

```
a = $00111111222222
b = $00000000000000
```

Procedure

```
Type: change a $00111111222222 b 0<return>
      asm p:$507 tfr a,b<return>
      change pc $507<return>
      step<return>
```

Results

```
a = $00111111222222
b = $00111111222222
```

Table 4.6 Transfer Data ALU Register, Negate Accumulator, Absolute Value, Normalize Accumulator Iteration, and Clear Accumulator Instructions

Mnemonic	Operation	Assembler Syntax	Source S	Destination D
TFR	$S \rightarrow D$	TFR S,D [PM]	A	B
			B	A
			X1,X0	A,B
			Y1,Y0	A,B
NEG	$0-D \rightarrow D$	NEG D [PM]		A,B
ABS	$ D \rightarrow D$	ABS D [PM]		A,B
NORM*	if $E.\bar{U}.Z=\bar{1}$, then	NORM Rn,D		A,B
	ASL $\rightarrow D$			
	$Rn-1 \rightarrow Rn$			
	Else if $E=1$, then			
	ASR $\rightarrow D$			
	$Rn+1 \rightarrow Rn$			
	Else			
	NOP			
CLEAR	$0 \rightarrow D$	CLR D [PM]		A,B

*Rn = Address register R0–R7

E = Extension bit (fifth bit of the status register)

U = Unnormalized bit (fourth bit of the status register)

Z = Zero bit (second bit of the status register)

EXAMPLE 4 – 2 4

Negate accumulator

Problem

Execute the instruction **NEG A** to subtract the A-Accumulator from zero and store the result in the A-Accumulator. Assume that the contents of the A-Accumulator before execution of the instruction are:

a = \$00333333333333

Procedure

```
Type: change a $00333333333333<return>
asm p:$508 neg a<return>
change pc $508<return>
step<return>
```

Result

a = \$ffcccccccccd

Note that the result can be checked by complementing each bit of the number \$003333333333 to form its one's complement \$ffcccccccccc, then adding one to the one's complement number to form the two's complement number \$ffcccccccccd.

E X A M P L E

4 - 2 5

Absolute value*Problem*

Execute the instruction **ABS A** to take the absolute value of the A-Accumulator and store the result in the A-Accumulator. Assume that the contents of the A-Accumulator before execution of the instruction are:

a = \$ff900000000000

Procedure

```
Type: change a $ff900000000000<return>
      asm p:$509 abs a<return>
      change pc $509<return>
      step<return>
```

Result

a = \$00700000000000

Note that the absolute value of the decimal number -0.875 (\$ff900000000000) is the decimal number 0.875 (\$00700000000000).

E X A M P L E

4 - 2 6

Normalize accumulator iteration*Problem*

Execute the instruction **NORM R0,B** to perform one iteration of the normalization of the B-Accumulator.

Accumulator Extension Bit is Set: Assume that the contents of the B-Accumulator, Address Register R0, and the Status Register (SR) before execution of the instruction are:

b = \$22888888000000
 b2 = \$22 b1 = \$888888 b0 = \$000000
 r0 = \$ffa
 sr = \$0020

Procedure

If the Extension Bit (bit-5) of the Status Register (SR) is set before execution of the instruction, then the B-Accumulator is arithmetically shifted right one bit position and the Address Register R0 is incremented by one.

```
Type: change a 0 b 0<return>
      change r0 $ffa sr $0020<return>
      change b $22888888000000<return>
```

```
asm p:$50a norm r0,b<return>
change pc $50a<return>
step<return>
```

Results

```
b = $11444444000000
b2 = $11 b1 = $444444 b0 = $000000
r0 = $fffb
sr = $00e0
```

Accumulator Extension is not in Use: Assume that the contents of the B accumulator register, the R0 address register, and the status register before execution of the instruction are:

```
b = $00111111000000
r0 = $ffff
sr = $0010
```

Procedure

If the Extension Bit (bit-5) of the Status Register (SR) is cleared, and the Unnormalized Bit (bit-4) is set, before execution of the instruction, then the B-Accumulator is arithmetically shifted left one bit position and the Address Register R0 is decremented by one.

```
Type: change a 0 b 0<return>
change r0 $ffff sr $0010<return>
change b $00111111000000<return>
asm p:$50b norm r0,b<return>
change pc $50b<return>
step<return>
```

Results

```
b = $00222222000000
r0 = $fffe
sr = $0090
```

E X A M P L E**4 – 2 7****Clear accumulator****Problem**

Execute the instruction **CLR A** to clear the A-Accumulator. Assume that the contents of the A-Accumulator before execution of the instruction are:

```
a = $11222222333333
```

Procedure

```
Type: change a $11222222333333<return>
asm p:$50c clr a<return>
change pc $50c<return>
step<return>
```

Result

```
a = $00000000000000
```

4.4.3 Logic Instructions

The Logic instructions use the Data ALU to perform all logic-type operations. Source operands for the Logic instructions, except ANDI and ORI, are contained in the Data ALU's Input Registers or Accumulators; destinations for the results generated by execution of the Logic instructions, except ANDI and ORI, are either the A-Accumulator or the B-Accumulator. The destination for the results generated by execution of the ANDI or ORI instructions is the Mode Register MR, Condition Code Register CCR, or Operating Mode Register OMR. The Logic instructions, except ANDI and ORI, allow up to two parallel-move operations to be executed concurrently with execution of the instruction's opcode. The parallel-move operations use the XD and YD Data Buses. As a result, the parallel-move operations can transfer new prefetched data from X- and/or Y-Memory to the Data ALU for use in subsequent instructions and can transfer the results generated by the execution of previous instructions from the Data ALU to X- and/or Y-Memory.

The Logic instructions execute in one instruction cycle and can affect all of the bits in the Condition Code Register. The Logic instructions are:

AND	logical AND
ANDI	AND immediate control register
EOR	Logical exclusive OR
LSL	Logical shift left
LSR	Logical shift right
NOT	Complement
OR	Logical inclusive OR
ORI	OR immediate control register
ROL	Rotate left
ROR	Rotate right

a. *Logic Instructions:* The Logic instructions **AND (AND)**, **logic inclusive OR (OR)**, **logic exclusive OR (EOR)**, and **logic complement NOT (NOT)** are summarized in Table 4.7. These instructions are 24-bit operations that are performed on bits 24–47 of the A-Accumulator or B-Accumulator.

b. *Shift and Rotate Logic Instructions:* The **logic-shift accumulator left (LSL)**, **logic-shift accumulator right (LSR)**, **rotate accumulator left (ROL)**, and **rotate accumulator right (ROR)** logic instructions are summarized in Table 4.8.

EXAMPLE 4 – 28

Logic-shift accumulator right

Problem

Execute the instruction **LSR A** to logic-shift bits 24–47 of the A-Accumulator one bit position to the right and store the result in the A-Accumulator. Assume that the contents of the A-Accumulator and the Status Register (SR) before execution of the instruction are:

Table 4.7 Logic Instructions

Mnemonic	Operation	Assembler Syntax	Source S	Destination D
AND	$S \cdot D \rightarrow D$	AND S,D [PM]	X1,X0	A,B
			Y1,Y0	A,B
OR	$S + D \rightarrow D$	OR S,D [PM]	X1,X0	A,B
			Y1,Y0	A,B
EOR	$S \oplus D \rightarrow D$	EOR S,D [PM]	X1,X0	A,B
			Y1,Y0	A,B
NOT	$\bar{D} \rightarrow D$	NOT D [PM]		A,B

Table 4.8 Shift and Rotate Logic Instructions

Mnemonic	Operation	Assembler Syntax	Destination D
LSL	Logical Shift Accumulator Left	LSL D [PM]	A,B
ROL	Rotate Accumulator Left	ROL D [PM]	A,B
LSR	Logical Shift Accumulator Right	LSR D [PM]	A,B
ROR	Rotate Accumulator Right	ROR D [PM]	A,B

a = \$00222222444444
 sr = \$0001

Procedure

The **logic-shift accumulator right** instruction **LSR A** is used to logic-shift bits 24–47 of the A-Accumulator one bit position to the right and store the result in the A-Accumulator. The remaining bits of the A-Accumulator are not affected. The Carry Bit (bit 0) of the SR receives the least significant bit (bit 24) that is shifted out of the A-Accumulator. A zero is shifted into bit-47 of the A-Accumulator.

```
Type: change a 0 b 0<return>
      change sr $0001<return>
      change a $00222222444444<return>
      asm p:$600 lsr a<return>
      change pc $600<return>
      step<return>
```

Results

a = \$00111111444444
sr = \$0080

Note that the Carry Bit (bit 0) of the SR receives the zero that is shifted out of bit-24 of the A-Accumulator. The Scaling Bit of the SR is set to detect the data growth.

E X A M P L E 4 – 2 9**Rotate accumulator left***Problem*

Execute the instruction **ROL A** to rotate bits 24–47 of the A-Accumulator one bit position to the left and store the results in the A-Accumulator. Assume that the contents of the A-Accumulator and the Status Register (SR) before execution of the instruction are:

a = \$00222222000000
sr = \$0001

Procedure

The **rotate accumulator left** instruction **ROL A** is used to rotate bits 24–47 of the A-Accumulator one bit position to the left and store the results in the A-Accumulator. The remaining bits of the A-Accumulator are not affected. The Carry Bit (bit 0) of the SR receives the bit that is shifted out of the bit-47 position of the A-Accumulator. The previous value of the carry bit is shifted into bit-24 of the A-Accumulator.

Type: change a 0 b 0<return>
change sr \$0001<return>
change a \$00222222000000 sr \$0001<return>
asm p:\$601 rol a<return>
change pc \$601<return>
step<return>

Results

a = \$0044444450000000
sr = \$0080

Note that the Carry Bit of the SR receives the *zero* that is shifted out of bit-47 of the A-Accumulator, and the previous value of the Carry Bit is shifted into bit-24 of the A-Accumulator. The Scaling Bit of the SR is set to detect the data growth.

c. Immediate Logic Instructions: The **AND immediate control register (ANDI)** and the **OR immediate control register (ORI)** logical instructions are summarized in Table 4.9.

Table 4.9 Immediate Logic Instructions

Mnemonic	Operation	Assembler Syntax	Destination D
ANDI	#xx.D → D	AND(I) #xx,D	MR,CCR,OMR
ORI	#xx+D → D	OR(I) #xx,D	MR,CCR,OMR

E X A M P L E 4 – 3 0

AND immediate to control register*Problem*

Execute the instruction **ANDI #f3,MR** to logic-AND the contents of the Mode Register (MR) with the 8-bit immediate operand \$f3 and store the results in the MR. Assume that the contents of the Status Register before execution of the instruction are:

sr = \$afff

Procedure

```
Type: change a 0 b 0<return>
      change sr $afff<return>
      asm p:$602 andi #f3,mr<return>
      change pc $602<return>
      step<return>
```

Result

sr = \$a3ff

4.4.4 Bit Manipulation Instructions

The bit manipulation instructions are:

BCLR	Bit test and clear
BSET	Bit test and set
BCHG	Bit test and change
BTST	Bit test on memory and registers

The bit manipulation instructions test the state of any single bit in a memory location or a register and then optionally set, clear, or invert the bit. The Carry Bit of the Condition Code Register will reflect the result of the bit test. The bit manipulation instructions are summarized in Table 4.10.

E X A M P L E 4 – 3 1

Bit test on memory*Problem*

Execute the instruction **BTST #2,x:\$200** to test bit-2 of X-Memory location \$200. The contents of X-Memory location \$200 and the Status Register before execution of the instruction are:

x:\$200 \$000004
sr = \$0000

Table 4.10 Bit Manipulation Instructions

Mnemonic	Operation	Assembler Syntax
BCLR*	$D(n) \rightarrow C$	BCLR #n,X:<ea>
	$0 \rightarrow D(n)$	BCLR #n,Y:<ea>
		BCLR #n,D
BSET*	$D(n) \rightarrow C$	BSET #n,X:<ea>
	$1 \rightarrow D(n)$	BSET #n,Y:<ea>
		BSET #n,D
BCHG*	$D(n) \rightarrow C$	BCHG #n,X:<ea>
	$\overline{D(n)} \rightarrow D(n)$	BCHG #n,Y:<ea>
		BCHG #n,D
BTST*	$D(n) \rightarrow C$	BTST #n,X:<ea>
		BTST #n,Y:<ea>
		BTST #n,D

*D(n) = nth bit of the 24-bit destination operand field.

0 ≤ n ≤ 23.

Procedure

The **bit test on memory** instruction **BTST #2,x:\$200** is used to test bit-2 of X-Memory location \$200. After the bit is tested, its state is reflected in the Carry Bit of the CCR.

```
Type: change a 0 b 0<return>
      change x:$200 $000004 sr $0000<return>
      display x:$200<return>
      asm p:$603 btst #2,x:$200<return>
      change pc $603<return>
      step<return>
```

Results

```
x:$200 $000004
sr = 0001
```

Note that the value of bit-2 of X-Memory location \$200 is one. Therefore, the Carry Bit of the CCR (bit 0 of SR) is set to a “one.”

4.4.5 Loop Instructions

The loop instructions are **DO** and **ENDDO**. The interruptible **DO** instruction initiates the beginning of a hardware **DO** loop. The **ENDDO** instruction can be used to terminate a hardware **DO** loop before that loop is completed. The contents of the

Loop Counter (LC) specify the number of times a hardware **DO** loop is to be repeated. The contents of the Loop Address (LA) Register indicate the location of the last instruction word in a hardware **DO** loop. The LC and LA can be pushed to the System Stack, thereby allowing hardware **DO** loops to be interrupted or nested; they can be unstacked by a **RET** instruction or an **ENDDO** instruction.

E X A M P L E 4 – 3 2

Problem

Execute the program **DOLOOP.CLD** to multiply the signed fraction data A(i) by the signed fraction data B(i), for i = 1, 2, and 3. The contents of X-Memory locations \$1000–\$1002, Y-Memory locations \$2000–\$2002, and X-Memory locations \$1500–\$1502 before execution of the program are:

```
x:$1000 $400000 $400000 $400000
x:$1500 $000000 $000000 $000000
y:$2000 $300000 $300000 $300000
```

Procedure

The signed fraction data A(1), A(2), A(3) and B(1), B(2), B(3) are stored in the X-Memory locations \$1000–\$1002 and Y-Memory locations \$2000–\$2002, respectively. The multiplication results are stored in X-Memory locations \$1500–\$1502. The **DOLOOP.ASM** assembler program is shown in Fig. 4.1.

1. Power-on and boot the PC.
2. Insert the DSP56002 Demonstration Software diskette into Drive “A.”
3. Load the Debug-EVM Software program by following steps 5–6 of section 1.4.
4. Select a second and a third data windows (data window #2 and data window #3) by clicking the mouse on menu, view, data2, data3, and done, respectively. Hit <alt-1> to select data window #1, <alt-2> to select data window #2, and <alt-3> to select data window #3.
5. Type:


```
change la 0 lc 0 <return>
change x:$1000..$1002 $400000<return>
change x:$1500..$1502 0<return>
change y:$2000..$2002 $300000<return>
display x:$1000<return>
<alt-2> (remember to select the command window with the mouse click)
display y:$2000<return>
<alt-3> (remember to select the command window with the mouse click)
display x:$1500<return>
load a:doloop.cld<return>
change pc $100<return>
step 6<return>
<alt-2>
<alt-1> (remember to select the command window with the mouse click)
step 11<return>
<alt-2>
<alt-3> (remember to select the command window with the mouse click)
```

The contents of the Loop Counter (LC), the Loop Address (LA) Register, X-Memory locations \$1000–\$1002, Y-Memory locations \$2000–\$2002, and X-Memory locations \$1500–\$1502 at the start and at the end of the DO LOOP are:

Start of Do loop	la = \$010d
	lc = \$0003
	x:\$1000 \$400000 \$400000 \$400000
	y:\$2000 \$300000 \$300000 \$300000
	x:\$1500 \$000000 \$000000 \$000000
End of DO loop	la = \$0000
	lc = \$0000
	x:\$1000 \$400000 \$400000 \$400000
	y:\$2000 \$300000 \$300000 \$300000
	x:\$1500 \$180000 \$180000 \$180000

Note that the result of multiplying 1/2 (\$400000) by 3/8 (\$300000) is 3/16 (\$180000). The DO instruction loads the LA register with the location of the last instruction word in the program loop (\$010D).

```

org      p:$100
move     #$1000,r0
move     #$2000,r4
move     #$1500,r1
move     x:(r0),x0
move     y:(r4),y0
do        #3,_end
mpyr     x0,y0,a      x:(r0)+,x0  y:(r4)+,y0
nop
move     a,x:(r1)+
nop
_end

```

Fig. 4.1 DOLOOPASM Assembler Program

EXAMPLE

4 – 3 3

Problem

Execute the program **DOLOOP1.CLD** to multiply the signed integer data A(i) by the signed integer data B(i), for i=1,2, and 3. The contents of X-Memory locations \$1000–\$1002, Y-Memory locations \$2000–\$2002, and L-Memory locations \$1500–\$1502 before execution of the program are:

x:\$1000	\$000002	\$000002	\$000002
y:\$2000	\$000138	\$000138	\$000138
l:\$1500	\$000000000000	\$000000000000	\$000000000000

Procedure

The signed integer data A(1), A(2), A(3) and B(1), B(2), B(3) are stored in X-Memory locations \$1000–\$1002 and Y-Memory locations \$2000–\$2002, respectively. When multiplying two signed integer numbers, the contents of the accumulator are right-shifted immediately after the multiplication to obtain the correct results. The right-shift operation shifts the least significant bit out of the accumulator and shifts a sign-extension bit in the most significant bit. The results of the multiplications are stored in L-Memory locations \$1500–\$1502. A list of the assembler program **DOLOOP1.ASM** is shown in Fig. 4.2.

```
Type: change la 0 lc 0<return>
      change x:$1000..$1002 $000002<return>
      change l:$1500..$1502 0<return>
      change y:$2000..$2002 $000138<return>
      display x:$1000<return>
      <alt-2> (remember to select the command window with the mouse click)
      display y:$2000<return>
      <alt-3> (remember to select the command window with the mouse click)
      display l:$1500<return>
      load a:doloop1.cld<return>
      change pc $100<return>
      break #1 p:$10f<return>
      go <return>
      <alt-2>
      <alt-1> (remember to select the command window with the mouse click)
```

Results

The contents of X-Memory locations \$1000–\$1002, Y-Memory locations \$2000–\$2002, and L-Memory locations \$1500–\$1502 after execution of the program are:

```
x:$1000 $000002 $000002 $000002
y:$2000 $000138 $000138 $000138
l:$1500 $000000000270 $000000000270 $000000000270
```

Note that the result of multiplying 2 (\$000002) by 312 (\$000138) is 624 (\$000000000270).

```
org      p:$100
move     #$1000,r0
move     #$2000,r4
move     #$1500,r1
move     x:(r0),x0
move     y:(r4),y0
do       #3,_end
mpy      x0,y0,a      x:(r0)+,x0      y:(r4)+,y0
nop
asr      a
move     a10,l:(r1)+
nop
_end
```

Handwritten note: Note the difference it is made

Fig. 4.2 DOLOOP1.ASM Assembler Program

4.4.6 Program Control Instructions

The Program Control instructions include jumps, conditional jumps, and other instructions that affect the PC and System Stack. Optional data transfers over the XDB and YDB may be specified in some of the program control instructions. These instructions may affect the Condition Code Register. The Program Control instructions are:

DEBUG	Enter debug mode
DEBUGcc	Enter debug mode conditionally
Ill	Illegal instruction
Jcc	Jump conditionally
JMP	Jump
JCLR	Jump if bit clear
JSET	Jump if bit set
JSc	Jump to subroutine conditionally
JSR	Jump to subroutine
JSCLR	Jump to subroutine if bit clear
JSSET	Jump to subroutine if bit set
NOP	No operation
REP	Repeat next instruction
RESET	Reset on-chip peripheral devices
RTI	Return from interrupt
RTS	Return from subroutine
STOP	Stop processing
SWI	Software interrupt
WAIT	Wait for interrupt

EXAMPLE 4 - 3 4

Repeat next instruction

Problem

Execute the two instructions **REP #3** and **ASR B** to repetitively execute the **ASR B** instruction three times. The **repeat next (REP)** instruction is summarized in Table 4.11. The contents of the B-Accumulator and the Loop Counter (LC) before execution of the two instructions are:

b = \$00888888000000 lc = \$0300

Procedure

Type: change b \$00888888000000 lc \$0300<return>
asm p:\$700 rep #3<return>

Table 4.11 Repeat Next Instruction

Mnemonic	Operation	Assembler Syntax
REP	LC → TEMP; #xxx → LC	REP #xxx
	Repeat Next Instruction Until LC = 1	REP X:<ea>
	TEMP → LC	REP Y:<ea>
		REP S

```
asm p:$701 asr b<return>
change pc $700<return>
step <return>
step <return>
step <return>
step <return>
```

Results

The contents of the B-Accumulator and the Loop Counter before and after the execution of the program and after the execution of each **REP** instruction are:

Before program execution	b = \$00888888000000 lc = \$0300
After 1 st REP instruction	b = \$00888888000000 lc = \$0003
After 2 nd REP instruction	b = \$00444444000000 lc = \$0002
After 3 rd REP instruction	b = \$00222222000000 lc = \$0001
After program execution	b = \$00111111000000 lc = \$0300

E X A M P L E 4 – 3 5

Problem

Execute the program **PR.CLD** to test bit-7 of X-Memory location \$100 and jump if the Carry Bit of the CCR is cleared. The **PR.ASM** assembler program is shown in Fig. 4.3.

```
wait      org      p:$200
          btst     #7,x:$100
          jcc      wait
          end
```

Fig. 4.3 PR.ASM Assembler Program

Procedure

The **PR.CLD** program is used to test the bit-7 of X-Memory location \$100. The state of the bit is reflected in the Carry Bit of the CCR. If the Carry Bit is cleared, then program execution continues at P-Memory location \$200; if the Carry Bit is set, the PC is incremented by one to point to P-Memory location \$203.

- a. Type: change a 0 b 0<return>
 change x:\$100 \$40 sr \$310<return>
 display x:\$100<return>
 load a:pr.cld<return>
 step 2<return>

Results

sr= \$0310
 P:\$0200 0B7027 000100 = BTST #\$7,X:>\$100

Note that the Carry Bit (bit 0 of SR) of the CCR is cleared. Therefore, program execution continues at the P-Memory location \$200.

- b. Type: change x:\$100 \$80<return>
 change pc \$200<return>
 step 2<return>

Results

sr= \$0311
 P:\$0203 000000 = NOP

Note that the Carry Bit (bit 0 of SR) of the CCR is set. Therefore, program execution continues at P-Memory location \$203.

The program control instructions shown in Table 4.12 test the state of any single bit in a memory location or a register and jumps (or jumps to subroutine) if the bit is "set" or "clear."

EXAMPLE 4 - 3 6

Jump if bit set

Problem

Execute the instruction **JSET #2,X:(R0),\$1000** to test bit-2 of the X-Memory location pointed to by Address Register R0. The contents of X-Memory location \$200, the Program Counter (PC), and Address Register R0 before execution of the instruction are:

x:\$200 \$000004
 pc = \$605
 r0 = \$0200

Procedure

The **jump if bit set** instruction **JSET #2,X:(R0),\$1000** is used to test bit-2 of the X-Memory location pointed to by the Address Register R0. If the bit is set, the instruction in P-Memory location \$1000 is fetched. Otherwise, the Program Counter is simply incremented to point to the next instruction.

Type: change x:\$200 \$000004 r0 \$200<return>
 asm p:\$605 jset #2,x:(r0),\$1000<return>
 change pc \$605<return>

Table 4.12 Group of Program Control Instructions

Mnemonic	Operation	Assembler Syntax
JSET	If $S(n)^* = 1$	JSET #n,X:<ea>,xxxx
	Then $xxxx \rightarrow PC$	JSET #n,Y:<ea>,xxxx
	Else $PC+1 \rightarrow PC$	JSET #n,S,xxxx
JCLR	If $S(n)^* = 0$	JCLR #n,X:<ea>,xxxx
	Then $xxxx \rightarrow PC$	JCLR #n,Y:<ea>,xxxx
	Else $PC+1 \rightarrow PC$	JCLR #n,S,xxxx
JSSET	If $S(n)^* = 1$	JSSET #n,X:<ea>,xxxx
	Then $SP+1 \rightarrow SP$	JSSET #n,Y:<ea>,xxxx
	$PC \rightarrow SSH$	JSSET #n,S,xxxx
	$SR \rightarrow SSL$	
	$xxxx \rightarrow PC$	
	Else $PC+1 \rightarrow PC$	
JSCLR	If $S(n)^* = 0$	JSCLR #n,X:<ea>,xxxx
	Then $SP+1 \rightarrow SP$	JSCLR #n,Y:<ea>,xxxx
	$PC \rightarrow SSH$	JSCLR #n,S,xxxx
	$SR \rightarrow SSL$	
	$xxxx \rightarrow PC$	
	Else $PC+1 \rightarrow PC$	

* $S(n) = n^{\text{th}}$ bit in the source operand.

```
display x:$200<return>
step<return>
```

Results

```
x:$200 $000004
pc = $1000
r0 = $0200
```

Note that bit-2 of the X-Memory location \$200 is “set.” Therefore, the instruction in P-Memory location \$1000 is fetched and the value of the PC is \$1000. If bit was “clear,” the PC would have been simply incremented from \$605 to \$607.

4.4.7 DSP56002 Instruction Set Summary

The DSP56002 instruction set is summarized in Table 4.13.

Note, this chapter covers most of the DSP56002 processor's instructions. For more information on the DSP56002 processor's instruction set, refer to Appendix A of the *DSP56000 Family Manual*.

Table 4.13 Instruction Set Summary

Move Instructions	
LUA	Load updated address
MOVE	Move data
MOVEC	Move control register
MOVEM	Move program memory
MOVEP	Move peripheral data
Arithmetic Instructions	
ABS	Absolute value
ADC	Add with carry
ADD	Add
ADDL	Shift left then add
ADDR	Shift right then add
ASL	Arithmetic shift left
ASR	Arithmetic shift right
CLR	Clear
CMP	Compare
CMPM	Compare magnitude
DEC	Decrement by one
DIV	Divide iteration
INC	Increment by one
MAC	Multiply/accumulate
MACR	Multiply/accumulate and round
MPY	Multiply
MPYR	Multiply and round
NEG	Negate
NORM	Normalize iteration
RND	Round

Table 4.13 Instruction Set Summary (Continued)

SBC	Subtract with carry
SUB	Subtract
SUBL	Shift left then subtract
SUBR	Shift right then subtract
Tcc	Transfer Conditionally
TFR	Transfer
TST	Test
Logical Instructions	
AND	logical AND
ANDI	AND immediate control register
EOR	Logical exclusive OR
LSL	Logical shift left
LSR	Logical shift right
NOT	Complement
OR	Logical inclusive OR
ORI	OR immediate control register
ROL	Rotate left
ROR	Rotate right
Bit Manipulation Instructions	
BCLR	Bit test and clear
BSET	Bit test and set
BCHG	Bit test and change
BTST	Bit test on memory
JCLR	Jump if bit clear
JSET	Jump if bit set
JSCLR	Jump to subroutine if bit clear
JSSET	Jump to subroutine if bit set
Loop Instructions	
DO	Start hardware loop
ENDDO	Exit from hardware loop

Table 4.13 Instruction Set Summary (Continued)

Program Control Instructions	
DEBUG	Enter debug mode
DEBUGcc	Enter debug mode conditionally
ILL	Illegal instruction
Jcc	Jump conditionally
JMP	Jump
JScc	Jump to subroutine conditionally
JSR	Jump to subroutine
NOP	No operation
REP	Repeat instruction
RESET	Reset on-chip peripheral devices
RTI	Return from interrupt
RTS	Return from subroutine
STOP	Stop instruction processing
SWI	Software interrupt
WAIT	Wait for interrupt

4.5 REFERENCE

1. Motorola Inc., *DSP56000 Digital Signal Processing Family Manual*, 1992.

Introduction to Digital Signal Processing Systems

In this chapter, a simple DSP system consisting of two analog-to-digital (A/D) converters, a DSP56002 processor, and two digital-to-analog (D/A) converters is implemented and exercised. A DSP56000EVM Evaluation Module (EVM) is used to provide and control the DSP56002 processor and a CS4215 Multimedia Audio Codec. The CS4215 multimedia audio codec has two channels of 16-bit A/D converters and two channels of D/A converters. The DSP56002 processor's Synchronous Serial Interface (SSI) port is used to accommodate serial data transfers from the two A/D converters to the DSP56002 processor and from the DSP56002 processor to the two D/A converters.

In section 5.2, the CS4215 multimedia audio codec is described followed by a discussion of the DSP56002 processor's instruction codes that are needed to set up and control the CS4215 audio codec in the control and data modes. Sections 5.3 and 5.4 describe the DSP56002 processor's instruction codes that are needed to:

- A. select the DSP56002 processor's Port C general purpose input/output pins needed to control the CS4215 multimedia audio codec
- B. select the DSP56002 processor's on-chip SSI port and set the SSI port's control registers to receive data from the two A/D converters and transmit data to the two D/A converters
- C. enable the SSI port's interrupts to receive data from the two A/D converters and transmit data to the two D/A converters

In section 5.5, the complete DSP system is then exercised and implemented.

5.1 DSP SYSTEM

In the DSP System defined above and shown in Figs. 5.1 and 5.2, two analog input signals (left and right signals) are digitized by the two A/D converters on the EVM. Each digitized signal is output from the corresponding A/D converter as 16-bit serial data stream delimited by a Frame Sync Output (FSO) signal. Via the receive channel of the DSP56002 processor's SSI port, the digitized signals are received by the DSP56002 processor. The DSP56002 processor routes the digitized signals from the receive chan-

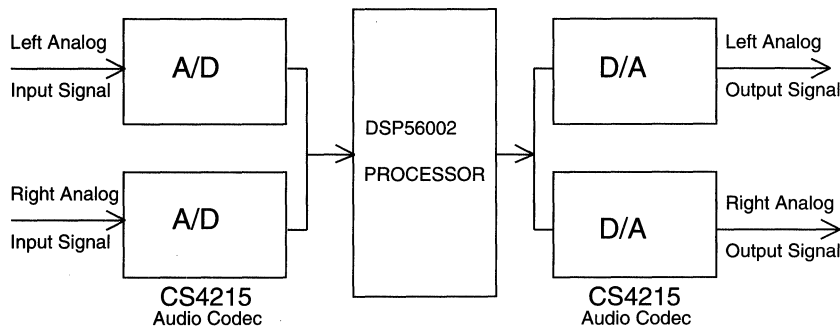


Fig. 5.1 DSP System

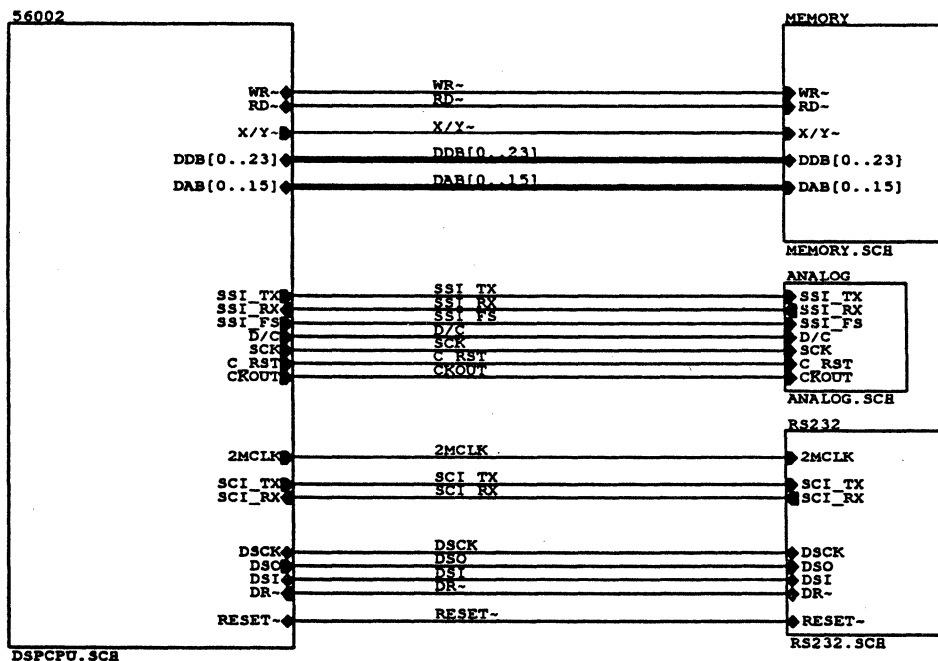


Fig. 5.2 EVM Block Diagram

nel of its SSI port to the transmit channel of its SSI port. The algorithm that the DSP56002 processor executes to route the digitized signals without modification is developed in section 5.5 of this chapter. (Note: in later chapters, this algorithm will be expanded to modify the digitized signals.) Via the transmit channel of the DSP56002 processor's SSI port, each digitized signal is output from the DSP56002 processor as a 16-bit serial data stream and routed to the serial input of the corresponding D/A converter. The two D/A converters construct the corresponding left and right analog output signals from the digitized left and right input signals.

In order for the D/A converter to be able to properly reconstruct the analog signal that was originally input to the A/D converter, the Nyquist Sampling Theorem must first be satisfied. That is, the analog signal input to the A/D converter must be sampled by the A/D converter at a rate greater than twice its highest frequency component (f_h). Sampling at this frequency ($2f_h$) is known as sampling at the Nyquist rate.

In practice, no practical analog input signal can be strictly band limited to the point where one can absolutely assure that the true Nyquist sampling rate is selected. Those high frequency components of the analog input signal that are greater than one-half the Nyquist sampling rate introduce an error component into the digitized signal that diminishes a D/A converter's ability to exactly reconstruct the original analog input signal. In general, the better the analog input signal is band limited, the lower the energy of those high frequency components that violate the Nyquist Sampling Theorem and the better the reconstruction of the original analog input signal.

Another error component is introduced by the D/A converter itself through its use of infinite duration sinc functions in its signal reconstruction process. Because of the accumulative effect of these various error components, exact reconstruction of the original analog input signal can only be approximated.

For the purposes of this book, if a practical analog input signal is sampled at the Nyquist rate, then a practical analog output signal can be reconstructed. If sampling of the input analog signal occurs below the Nyquist rate, then a practical analog output signal cannot be reconstructed. The effect of undersampling an analog input signal, i.e., sampling below the Nyquist rate, is known as "aliasing."

5.2 CS4215 MULTIMEDIA AUDIO CODEC

The CS4215 is a single-chip, stereo, CMOS multimedia codec that supports CD-quality music, FM radio-quality music, telephone-quality speech, and modems. The CS4215 has two channels of 16-bit analog-to-digital conversion and two channels of 16-bit digital-to-analog conversion. Both the ADCs and the DACs are delta-sigma converters. The features of the CS4215 multimedia audio codec include:

- ☛ sample frequencies from 4 KHz to 50 KHz
- ☛ 16-bit linear, 8-bit linear, μ -law, or A-law audio data coding
- ☛ programmable gain for analog inputs
- ☛ on-chip oscillators
- ☛ +5V power supply

- ☛ microphone and line level analog inputs
- ☛ headphone, speaker, and line outputs
- ☛ on-chip antialiasing/smoothing filters
- ☛ serial digital interface

Control for the functions available on the CS4215 audio codec, as well as the audio data, are communicated over the serial interface. The DSP56002 processor's Synchronous Serial Interface (SSI) is used to accommodate serial data transfers from the two A/D converters to the DSP56002 processor and from the DSP56002 processor to the two D/A converters. Figure 5.3 shows the connection of the CS4215 audio codec to the DSP56002 processor's Port C. The DSP56002 processor's Port C provides two control signals to the CS4215 audio codec: Active Low Reset Input (RESET~) and Data/Control Select Input (D/C~). The serial interface format has a variable number of time slots, depending on the number of CS4215 audio codecs attached to the bus. All CS4215 time slots have 8 bits. Each CS4215 requires 8 time slots (64 bits) to communicate all data. The CS4215 multimedia audio codec has the modes of operation shown in Fig. 5.3.

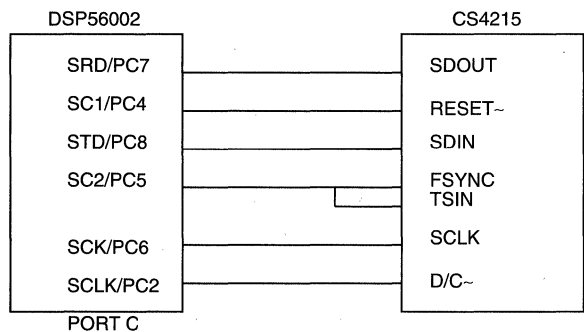


Fig. 5.3 Connection between the DSP56002 Processor's Port C and the CS4215 Audio Codec

5.2.1 Control Mode

The Control Mode is used to set up the CS4215 for subsequent operation in Data Mode by loading the internal control registers. Control mode is asserted by bringing D/C~ low. The information on SDIN and SDOUT pins of the CS4215 is the control information when D/C~ is low. The CS4215 control registers have the functions and time slot assignments shown in Table 5.1. The internal control registers and control

Table 5.1 Control Registers

CS4215 Time Slot	Description	CS4215 Time Slot	Description
1	Status	5	Parallel Port
2	Data Format	6	Reserved
3	Serial Port Control	7	Revision
4	Test	8	Reserved

time slots are shown in Fig. 5.4. The data listed below the register is its reset state. The register address is the time slot number when D/C~ is 0. The frame sync rate is equal to the value of the conversion frequency set by the DFR2-DFR0 bits of the Data

Control Time Slot 1, Status Register

	D7	D6	D5	D4	D3	D2	D1	D0
Register	0	0	1	MLB	OLB	CLB	RSRV	
Reset (R)	0	0	1	0	0	1	X	X

BIT	NAME	VALUE	FUNCTION
RSRV	Reserved Bits		Must be written as 0.
CLB	Control Latch Bit	1	Ensures proper transition between control and data mode.
OLB	Output Level Bit	0	Line full scale outputs are 2.8 Vpp (1Vrms) Headphone full scale output is 4.0 Vpp. Speaker full scale output is 8.0 Vpp.
		1	Line and Headphone full scale outputs are 2.0 Vpp. Speaker full scale output is 4.0 Vpp.
MLB	Microphone Level	0	20 dB Fixed Gain Enabled Full scale microphone inputs are 0.288 Vpp.
		1	20 dB Fixed Gain Disabled Full scale inputs are 2.88 Vpp.

Control Time Slot 2, Data Format Register

	D7	D6	D5	D4	D3	D2	D1	D0
Register	HPF	RSRV	DFR2	DFR1	DFR0	ST	DF1	DF0
Reset (R)	0	X	0	0	0	0	0	1

BIT	NAME	VALUE	FUNCTION
DF1-0	Data Format Selection	0 0 0 0 1 1 1 0 2 1 1 3	16-bit 2 ^S -complement linear. 8-bit μ -Law. 8-bit A-Law. 8-bit unsigned linear.
ST	Stereo Bit	0 1	Mono Mode. Stereo Mode.
DFR2-0	Data Conversion Frequency Selection	0 0 0 0 0 0 1 1 0 1 0 2 0 1 1 3 1 0 0 4 1 0 1 5 1 1 0 6 1 1 1 7	XTAL1(kHz) XTAL2 (kHz) CLKIN (+) 24.576 MHz 16.9344 MHz 3072 8 5.5125 1536 16 11.025 896 27.42857 18.9 768 32 22.05 448 NA 37.8 384 NA 44.1 512 48 33.075 2560 9.6 6.615
RSRV	Reserved Bit		Must be written as 0
HPF	High Pass Filter	0 1	Disabled. Enabled. A Digital High Pass Filter is used to force the ADC DC offset to zero.

Fig. 5.4 Internal Control Registers (Courtesy of Crystal Semiconductor Corp.)

Control Time Slot 3, Serial Port Control Register

	D7	D6	D5	D4	D3	D2	D1	D0
Register	ITS	MCK2	MCK1	MCK0	BSEL1	BSEL0	XCLK	XEN
Reset (R)	0	0	0	0	1	0	0	1

BIT	NAME	VALUE	FUNCTION
XEN	Transmitter Enable	0	Enable the serial data output.
		1	Disable (high-impedance state) serial data output.
XCLK	Transmit Clock	0	Receive SCLK and FSYNC from external source SLAVE Mode
		1	Generate SCLK and FSYNC MASTER Mode
BSEL1-0	Select Bit Rate	0 0	64 bits per frame.
		0 1	128 bits per frame.
		1 0	256 bits per frame.
		1 1	Reserved.
MCK2-0	Clock Source Select	0 0 0	SCLK is master clock, 256 bits per frame. BSEL must equal 2, and XCLK must equal 0.
		0 0 1	XTAL1, 24.576 MHz, is clock source.
		0 1 0	XTAL2, 16.9344 MHz, is clock source.
		0 1 1	CLKIN is clock source, and must be 256xFs.
ITS	Immediate Three-State	0	SCLK and FSYNC three-state up to 12 clocks after D/C goes low.
		1	SCLK and FSYNC three-state immediately after D/C goes low.

Control Time Slot 4, Test Register

	D7	D6	D5	D4	D3	D2	D1	D0
Register	TEST						ENL	DAD
Reset (R)	0	0	0	0	0	0	0	0

BIT	NAME	VALUE	FUNCTION
DAD	Loopback Mode	0	Digital-Digital Loopback.
		1	Digital-Analog-Digital Loopback.
ENL	Enable Loopback Testing	0	Disable.
		1	Enable.
TEST	Test bits		The TEST bits must be written as zero, otherwise special factory test modes may be invoked.

Control Time Slot 5, Parallel Port Register

	D7	D6	D5	D4	D3	D2	D1	D0
Register	PIO1	PIO0	RSRV					
Reset (R)	1	1	X	X	X	X	X	X

BIT	NAME	VALUE	FUNCTION
RSRV	Reserved Bits		Must be written as 0.
PIO1-0	Parallel I/O Bits	1 1	3 See the Parallel Input/Output Section.

Fig. 5.4 Internal Control Registers (Continued)

Control Time Slot 6, Reserved Register

	D7	D6	D5	D4	D3	D2	D1	D0
Register	RSRV							
Reset (R)	X	X	X	X	X	X	X	X

BIT	NAME	VALUE	FUNCTION
RSRV	Reserved Bits		Must be written as 0.

Control Time Slot 7, Version Register

	D7	D6	D5	D4	D3	D2	D1	D0
Register	RSRV				VER3	VER2	VER1	VER0
Reset (R)	X	X	X	X	0	0	1	0

BIT	NAME	VALUE	FUNCTION
VER3-0	Device Version Number	0 0 0 0 0 0 0 0 1 1 0 0 1 0 2 R	"C". See Appendix A. "D". See Appendix A. "E". This Data Sheet
RSRV	Reserved Bits		Must be written as 0.

Control Time Slot 8, Reserved Register

	D7	D6	D5	D4	D3	D2	D1	D0
Register	RSRV							
Reset (R)	X	X	X	X	X	X	X	X

BIT	NAME	VALUE	FUNCTION
RSRV	Reserved Bits		Must be written as 0.

Fig. 5.4 Internal Control Registers (Continued)

Format Register shown in Fig. 5.4. Each frame has either 64-, 128-, or 256-bit times depending on the BSEL bits in the Serial Port Control Register shown in Fig. 5.4. All CS4215 time slots contain 8 bits. The CS4215 supports four audio data formats: 16-bit 2's-complement linear, 8-bit unsigned linear, 8-bit A-Law, and 8-bit μ -law.

The following DSP56002 instruction codes are used to load a transmit buffer with the contents of the internal control registers needed to set up the CS4215 for subsequent operation in data mode. The Microphone Level Bit (MLB) is set to choose full-scale inputs of 2.88 peak to peak voltage. The Output Level Bit (OLB) is cleared to choose line scale outputs of 2.8 peak-to-peak voltage, a headphone full-scale output of 4.0 peak-to-peak voltage, and a speaker full-scale output of 8.0 peak-to-peak voltage. The Control Latch Bit (CLB) is initially cleared, then it is set to ensure a proper transition between control and data mode. The High Pass Filter (HPF) bit is cleared to disable the high pass filter used to force the ADC DC offset to zero. The two Data

Conversion Frequency (DFR) selection bits, DFR2 and DFR1, are set, and the DFR selection bit DFR0 is cleared to select a sampling frequency of 48 KHz. The Stereo Bit (ST) is set to choose the stereo mode. The two Data Format (DF) selection bits, DF1 and DF0, are cleared to choose a 16-bit 2's-complement linear data format. The Immediate Three-State (ITS) bit is set so the SCLK and FSYNC will be three-state immediately after D/C~ goes low. The two Master Clock (MCK) source select bits, MCK2 and MCK0, are cleared, and the MCK select bit, MCK1, is set to choose XTAL2 as a clock source. The Select Bit (BSEL) rate bits, BSEL1 and BSEL0, are cleared to select 64 bits per frame. The Transmitter Enable (XEN) bit is cleared to enable the serial data output. The Transmit Clock (XCLK) bit is set to generate SCLK and FSYNC (Master Mode).

```

NO_PREAMP      equ      $100000
SAMP_RATE_48   equ      $003000
STEREO         equ      $000400
DATA_16        equ      $200000

IMMED_3STATE   equ      $800000
XTAL2_SELECT   equ      $200000
BITS_64        equ      $000000
CODEC_MASTER    equ      $020000

CTRL_WD_12      equ      NO_PREAMP+SAMP_RATE_48+STEREO+DATA_16;CLB=0
CTRL_WD_34      equ      IMMED_3STATE+XTAL2_SELECT+BITS_64+CODEC_MASTER
CTRL_WD_56      equ      $000000
CTRL_WD_78      equ      $000000

```

```

org      x:0

```

```

RX_BUFF_BASE    equ      *
RX_data_1_2     ds       1      ;data time slot 1/2 for RX ISR
RX_data_3_4     ds       1      ;data time slot 3/4 for RX ISR
RX_data_5_6     ds       1      ;data time slot 5/6 for RX ISR
RX_data_7_8     ds       1      ;data time slot 7/8 for RX ISR

TX_BUFF_BASE    equ      *
TX_data_1_2     ds       1      ;data time slot 1/2 for TX ISR
TX_data_3_4     ds       1      ;data time slot 3/4 for TX ISR
TX_data_5_6     ds       1      ;data time slot 5/6 for TX ISR
TX_data_7_8     ds       1      ;data time slot 7/8 for TX ISR

SSI_FLAG        ds       1      ; bit flags used for word index
RX_PTR          ds       1      ; Pointer for rx buffer
TX_PTR          ds       1      ; Pointer for tx buffer

```

```

> ; - set up buffer with control mode data

```

```

move      #CTRL_WD_12,x0
move      x0,x:TX_BUFF_BASE
move      #CTRL_WD_34,x0
move      x0,x:TX_BUFF_BASE+1
move      #CTRL_WD_56,x0
move      x0,x:TX_BUFF_BASE+2
move      #CTRL_WD_78,x0
move      x0,x:TX_BUFF_BASE+3

```

5.2.2 Data Mode

The data mode is used during conversions to pass digital data between the CS4215 and the DSP56002 processor. Control of gain, attenuation, input selection, and output muting are embedded in the data stream. All CS4215 time slots contain 8 bits. The MSB of the data is transmitted/received first. The CS4215 data registers have the functions and time slot assignments shown in Table 5.2. Data time slots 1 and 2 contain audio data for the left channel. Data time slots 3 and 4 contain audio data for the right channel. In mono modes, only the left channel data is used. In 8-bit modes, only time slots 1 and 3 are used for the data. Data time slots 5 and 6 contain the output setting. Data time slots 7 and 8 contain the input setting. Figure 5.5 shows the data time slots 5, 6, 7, and 8.

Table 5.2 Data Registers

CS4215 Time Slot	Description	CS4215 Time Slot	Description
1	Left Audio MS8 bits	5	Output Setting
2	Left Audio LS8 bits	6	Output Setting
3	Right Audio MS8 bits	7	Input Setting
4	Right Audio LS8 bits	8	Input Setting

Data Time Slot 5, Output Setting

	D7	D6	D5	D4	D3	D2	D1	D0
Register	HE	LE	LO5	LO4	LO3	LO2	LO1	LO0
Reset (R)	0	0	1	1	1	1	1	1

BIT	NAME	VALUE	FUNCTION
LO5-0	Left Channel Output Attenuation Setting	1 1 1 1 1 1 63 R	1.5dB attenuation steps. LO5 is the MSB. 0 = no attenuation. 111111 = -94.5dB
LE	Line Output Enable	0 1	R Analog line outputs off (muted). Analog line outputs on.
HE	Headphone Output Enable	0 1	R Headphone output off (muted). Headphone output on.

Data Time Slot 6, Output Setting

	D7	D6	D5	D4	D3	D2	D1	D0
Register	ADI	SE	RO5	RO4	RO3	RO2	RO1	RO0
Reset (R)	1	0	1	1	1	1	1	1

BIT	NAME	VALUE	FUNCTION
RO5-0	Right Channel Output Attenuation Setting	1 1 1 1 1 1 63 R	1.5dB attenuation steps. RO5 is the MSB. 0 = no attenuation. 111111 = -94.5dB Not used in mono modes.
SE	Speaker Enable	0 1	R Speaker off (muted). Speaker on.
ADI	A/D Data Invalid	0 1	R A/D data valid. A/D data invalid. Busy in calibration.

Fig. 5.5 Data Time Slots 5, 6, 7, and 8 (Courtesy of Crystal Semiconductor Corp.)

Data Time Slot 7, Input Setting

	D7	D6	D5	D4	D3	D2	D1	D0
Register	PIO1	PIO0	OVR	IS	LG3	LG2	LG1	LG0
Reset (R)	1	1	0	0	0	0	0	0

BIT	NAME	VALUE		FUNCTION
LG3-0	Left Channel Input Gain Setting	0 0 0 0	R	1.5dB gain steps. LG3 is the MSB. 0 = no gain, 1111 = 22.5dB gain.
IS	Input Select	0 1	R	Line level inputs (LINL, LINR). Microphone level inputs (MINL, MINR).
OVR	Overrange	0	R	When read as 1, this bit indicates that an input over-range condition has occurred. The bit remains set until cleared by writing 0 into the register. Writing a 1 enables the overrange detection. The bit will remain 0 until an over-range occurs. Serial port clear has priority over internal settings.
PIO1-0	Parallel I/O	1 1 3	R	Parallel input/output bits.

Data Time Slot 8, Input Setting

	D7	D6	D5	D4	D3	D2	D1	D0
Register	MA3	MA2	MA1	MA0	RG3	RG2	RG1	RG0
Reset (R)	1	1	1	1	0	0	0	0

BIT	NAME	VALUE		FUNCTION
RG3-0	Right Channel Input Gain Setting	0 0 0 0	R	1.5dB gain steps. RG3 is the MSB. 0 = no gain, 1111 = 22.5dB gain.
MA3-0	Monitor Path Attenuation	1 1 1 1 15	R	6dB attenuation steps. MA3 is the MSB. 0 = no attenuation, 1111 = mute.

Fig. 5.5 Data Time Slots 5, 6, 7, and 8 (Continued)

The following DSP56002 instruction codes set up the transmit and receive buffers and provide the chosen output and input settings.

- A. Output Settings:** The Headphone Output Enable (HE) bit is set to enable the headphone output. The Line Output Enable (LE) bit is set to enable the analog line outputs. The Speaker Enable (SE) bit is cleared to mute the speaker. The A/D Data Invalid (ADI) bit is clear for the validation of the A/D data. The attenuation of the left and right channel outputs can also be set by writing the six least significant bits of the fifth and sixth data time slots, respectively (0 = no attenuation, 111111 = -94.5 dB, 63 steps with 1.5 dB attenuation per step).
- B. Input Settings:** The Monitor Path Attenuation (MA) bits MA3, MA2, MA1, and MA0 are set for the mute option (1111 = mute, 0 = no attenuation, 6 dB attenuation steps). The Input Select (IS) bit is set to select microphone level inputs instead of line level inputs. The gains of left and right channel inputs can be set by writing the four least significant bits of the seventh and eighth time slots, respectively (0 = no gain, 1111 = 22.5 dB gain, 15 steps with 1.5 dB per step).

```

HEADPHONE_EN    equ    $800000
LINEOUT_EN      equ    $400000
LEFT_ATTN       equ    $010000    ;63*LEFT_ATTN    = -94.5 dB, 1.5 dB steps

MIC_IN_SELECT   equ    $100000
MONITOR_ATTN    equ    $001000    ;15*MONITOR_ATTN = mute, 6 dB steps
RIGHT_ATTN      equ    $000100    ;63*RIGHT_ATTN  = -94.5 dB, 1.5 dB steps

OUTPUT_SET      equ    HEADPHONE_EN+LINEOUT_EN+(LEFT_ATTN*4)
INPUT_SET       equ    MIC_IN_SELECT+(15*MONITOR_ATTN)+(RIGHT_ATTN*4)

        move    #3,m0                ; Modulo 4 buffer.
        move    #3,m7                ; Modulo 4 buffer.
        move    #RX_BUFF_BASE,x0
        move    x0,x:RX_PTR          ; Initialize the rx pointer
        move    x:RX_PTR,r7          ; Load the pointer to the rx buffer.
        move    #TX_BUFF_BASE+1,x0
        move    x0,x:TX_PTR          ; Initialize the tx pointer
        move    x:TX_PTR,r0          ; Load the pointer to the tx buffer.
        move    #OUTPUT_SET,y0       ;headphones, line out, mute spkr, no attn
        move    y0,x:TX_BUFF_BASE+2
        move    #INPUT_SET,y0        ;no input gain, monitor mute
        move    y0,x:TX_BUFF_BASE+3

```

5.3 DSP56002 PROCESSOR PORT C

As shown in Fig. 5.6, the DSP56002 digital signal processor has:

- A.** a 50-pin memory expansion Port A
- B.** a 15-pin Port B that can be operated either as a general purpose I/O port or as a parallel Host Interface on-chip peripheral port
- C.** a 9-pin Port C where each individual pin can be selected to operate as a general purpose I/O pin or as an on-chip peripheral function pin. The on-chip peripherals include a Serial Communication Interface (SCI) port and a Synchronous Serial Interface (SSI) port. Any mix of general purpose I/O pins or on-chip peripheral function pins can be selected. When configured as general-purpose I/O, Port C can be used for device control. When configured as serial interfaces, Port C provides a convenient connection to digital-to-analog and analog-to-digital converters, other DSPs, processors, codecs, and any of several transducers

Each of the Host, SCI, and SSI on-chip peripherals has its own control, status, data, etc., registers. The DSP56002 processor accesses each of these registers as if they were memory-mapped I/O. These on-chip memory-mapped registers are accessed at X-Memory locations \$FFC0-\$FFFF as shown in Fig. 5.7.

5.3.1 General-Purpose I/O (GPIO)

Each Port C pin may be individually programmed as a GPIO pin or as a dedicated on-chip peripheral pin under software control. Port C GPIO control is shown in Fig. 5.8. When configured as GPIO, Port C pins are controlled by three memory-

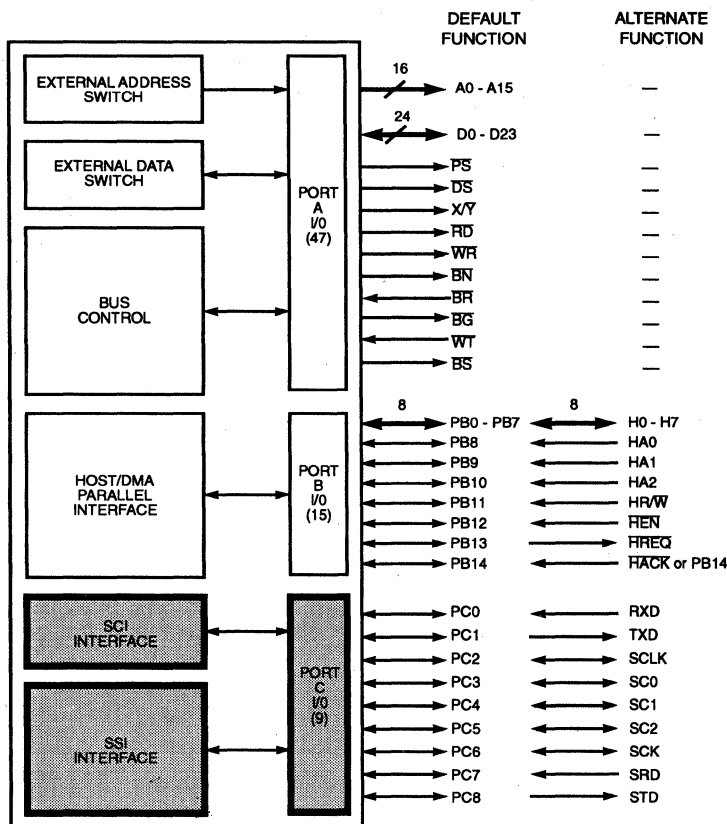


Fig. 5.6 DSP56002 Input/Output Diagram

mapped registers. These registers are the Port C Control Register (PCC), Port C Data Direction Register (PCDDR), and Port C Data Register (PCD). Port C GPIO registers are shown in Fig. 5.9. Pin selection between GPIO and SCI or SSI is made by setting the appropriate PCC bit (memory location X:\$FFE1) to zero for GPIO or to one for serial interface. The PCDDR (memory location X:\$FFE3) programs each pin corresponding to a bit in the PCD (memory location X:\$FFE5) as an input pin (if PCDDR = 0) or as an output pin (if PCDDR = 1). In Fig. 5.3, pins PC2 and PC4 of Port C are configured as output GPIO pins. If a pin is configured as a GPIO output and the processor writes to the PCD, the data will appear on the pin during the following instruction cycle. If the processor reads the PCD, the processor sees the contents of the PCD rather than the logic level on the pin, which allows the PCD to be used as a general purpose 15-bit register.

The following DSP56002 instruction codes program pins PC2 and PC4 of Port C as output GPIO pins and writes a zero to the PCD. PC4 generates the reset signal (RESET~) for the CS4215 audio codec and holds it low for 50 ms. PC2 brings D/C~ low to enter the control mode when RESET~ is low.

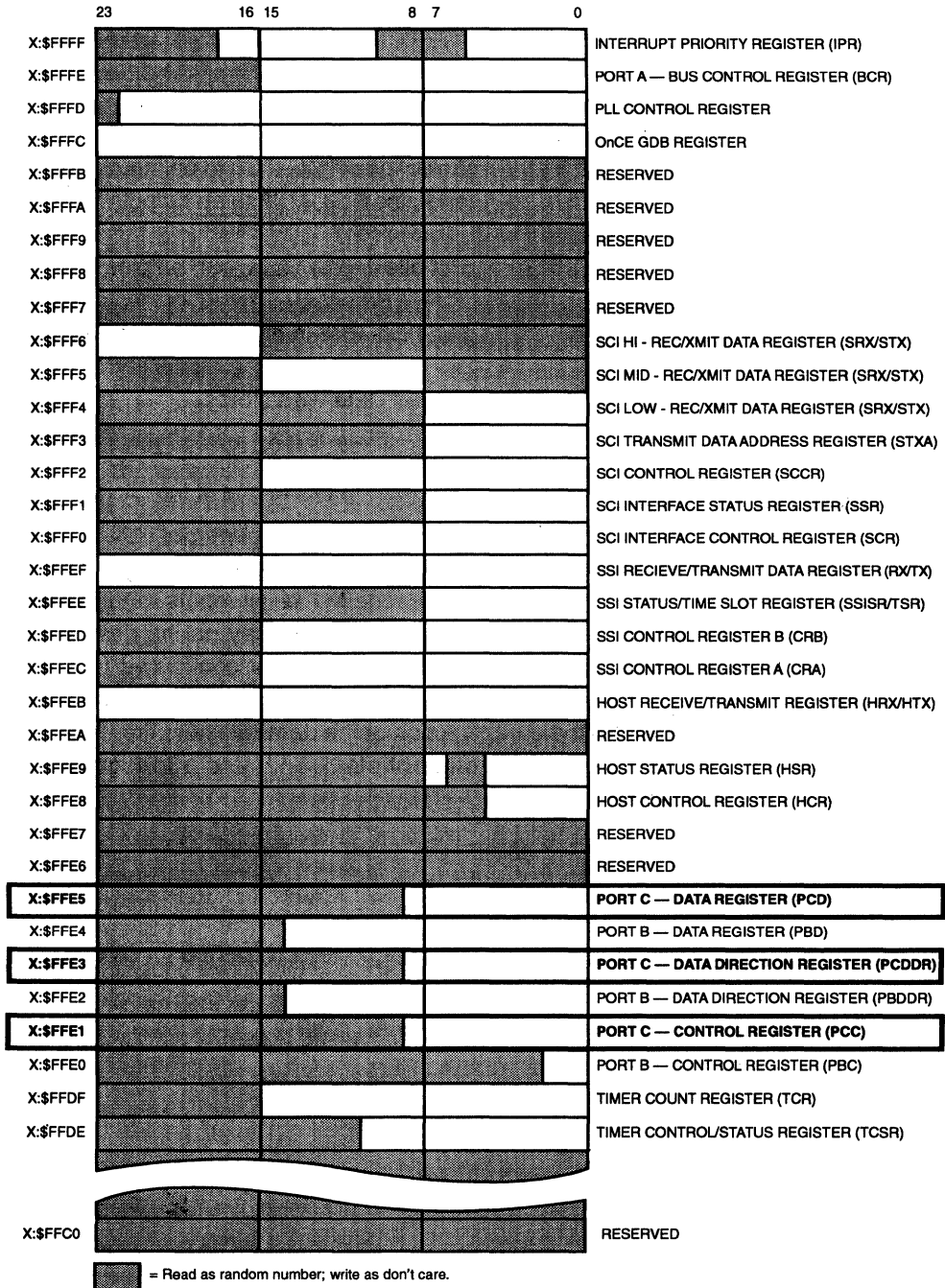


Fig. 5.7 On-chip Peripheral Memory


```

PCD      equ      $FFE5
PCDDR    equ      $FFE3
PCC      equ      $FFE1

        movep     #$0000,x:PCC      ; turn off ssi port
        movep     #$14,x:PCDDR     ; setup pc2 and pc4 as outputs
        movep     #$0,x:PCD        ; D/C~ and RESET~ = 0 ==> control mode

;--reset delay for codec --

```

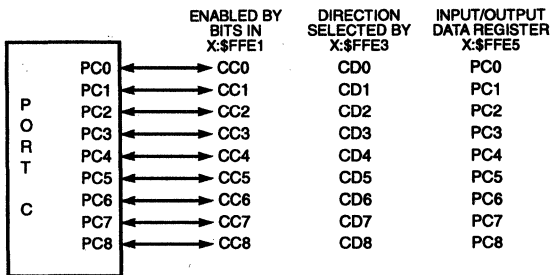
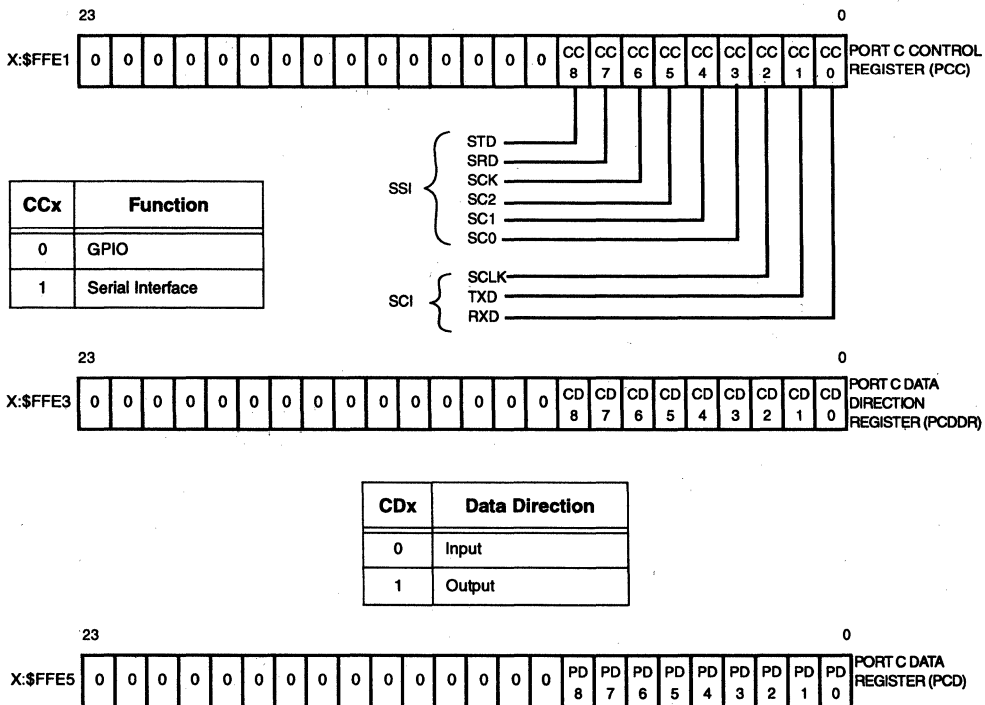


Fig. 5.8 Port C GPIO Control



NOTE: Hardware and software reset clears PCC and PCDDR.

Fig. 5.9 Port C GPIO Registers

```

do      #500,_delay_loop    ; 100 us * 500 = 50 ms
rep     #2000                ; 100 us delay
nop
_delay_loop
bset    #4,x:PCD              ; RESET~ = 1

```

5.3.2 DSP56002 Processor SSI Port Pins

The SSI port has six dedicated pins as shown in Fig. 5.10:

1. Serial Transmit Data (STD),
2. Serial Receive Data (SRD),
3. Serial Clock (SCK),
4. Serial Control (SC0),
5. Serial Control (SC1), and
6. Serial Control (SC2).

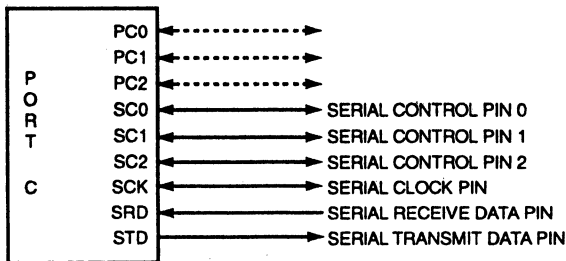


Fig. 5.10 Synchronous Serial Interface Pins

In Fig. 5.3, the SSI port's STD, SRD, SCK, and SC2 pins are used to interface the DSP56002 processor to the two A/D converters and two D/A converters. The STD pin is used to output data from the SSI port's Serial Transmit Shift Register. This data is routed to the input of the two D/A converters through the Serial Data In (SDIN) pin of the CS4215 audio codec. The SRD pin is used to receive data into the SSI port's Receive Data Shift Register. This data comes from the two A/D converters through the Serial Data Out (SDOUT) pin of the CS4215 audio codec. The SCK pin is used by the SSI port to accept an external clock during the data mode or output an internal clock during the control mode. The SCK pin of the DSP56002 processor is connected to the Serial Port Clock (SCLK) pin of the CS4215 audio codec. The SC2 pin of the DSP56002 processor is used by the SSI port to accept an external frame sync during the data mode or generate an internal sync during the control mode. The SC2 pin of the DSP56002 processor is connected to the Frame Sync (FSYNC) pin of the CS4215 audio codec.

5.3.3 SSI Port Selection

In the DSP System, the SSI port of the DSP56002 processor is used to receive serial data from the two A/D converters and transmit serial data to the two D/A converters. The following DSP56002 processor instruction code selects Port C to operate

as a SSI port by setting bits 8, 7, 6, 5, and 3 of the PCC register. The PCC register is shown in Fig. 5.9.

```
PCC      equ      $FFE1
         movep    #$01E8,x:PCC      ; Turn on ssi port
```

5.3.4 SSI Port Control Register “A” (CRA)

The operation of the SSI port is directed by two 16-bit read/write Control Registers CRA and CRB. The SSI registers are shown in Figs. 5.11 and 5.12. CRA controls the SSI port's:

- A. clock generator bit and frame sync rates
- B. word length
- C. number of words per frame for the serial data

The WL0-WL1 bits select the word length. The data word length can be 8, 12, 16, or 24 bits. The PM0-PM7 bits specify the divide ratio of the prescaler divider in the SSI clock generator. A divide ratio from 1 to 256 may be selected. The DC4-DC0 bits control the divide ratio for the programmable frame rate dividers used to generate the frame clocks. In the network, this ratio may be interpreted as the number of words per frame minus one. In normal mode, this ratio determines the word transfer rate. The divide ratio may range from 1 to 32 for normal mode and 2 to 32 for network mode. The PSR controls a fixed divide-by-eight prescaler in series with the variable prescaler. When PSR is cleared, the fixed prescaler is bypassed. When PSR is set, the fixed divide-by-eight prescaler is operational. The generation of the internal bit clock from the crystal clock frequency is shown in Fig. 5.13. CRA occupies X-Memory location \$FFEC.

In the control mode, the DSP56002 instruction codes shown below set the CRA control register to allow the DSP56002 processor to generate the external serial clock and external frame sync signal for the CS4215 audio codec where:

- A. CRA bit 15 is cleared (PSR = 0) to bypass the fixed prescaler (divide by one).
- B. CRA bits 0 and 1 are set (PW0 = PW1 = 1) and CRA bits 2, 3, 4, 5, 6 and 7 are cleared (PW2 = PW3 = PW4 = PW5 = PW6 = PW7 = 0) to generate a serial bit clock output equal to 2.5 MHz ($40 \text{ MHz} / [2 * 4 * 2]$) at the Serial Clock (SCK) pin.
- C. CRA bit 14 is set (WL1 = 1) and CRA bit 13 is cleared (WL0 = 0) to select a 16-bit word length.
- D. CRA bits 8 and 9 are set (DC0 = DC1 = 1) and CRA bits 10, 11, and 12 are cleared (DC2 = DC3 = DC4 = 0) to select four words per frame in the network mode. The network mode will be selected by the CRB Control Register.

```
CRA      equ      $FFEC
         movep    #$4303,x:CRA      ;40MHz/16 = 2.5MHz SCLK, WL=16 bits, 4W/F
```

In the data mode, the same DSP56002 instruction codes shown above set the CRA control register to allow the DSP56002 processor to receive an external serial

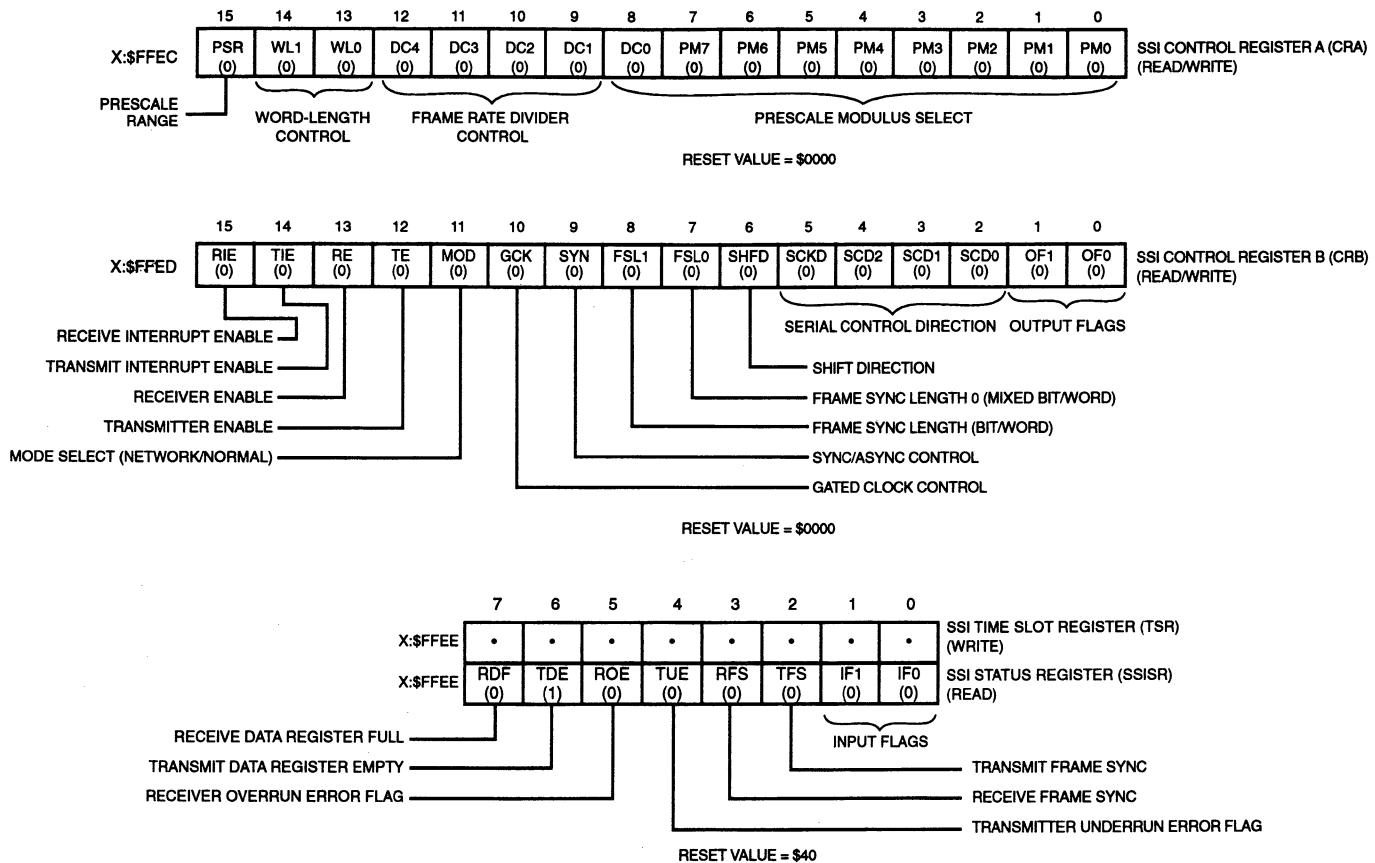
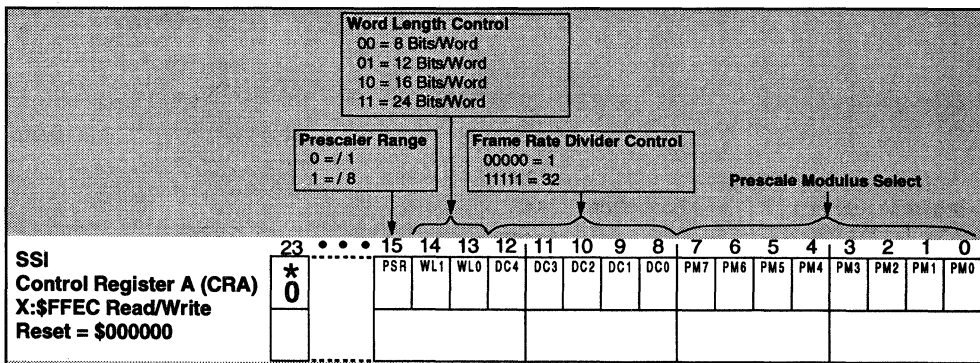
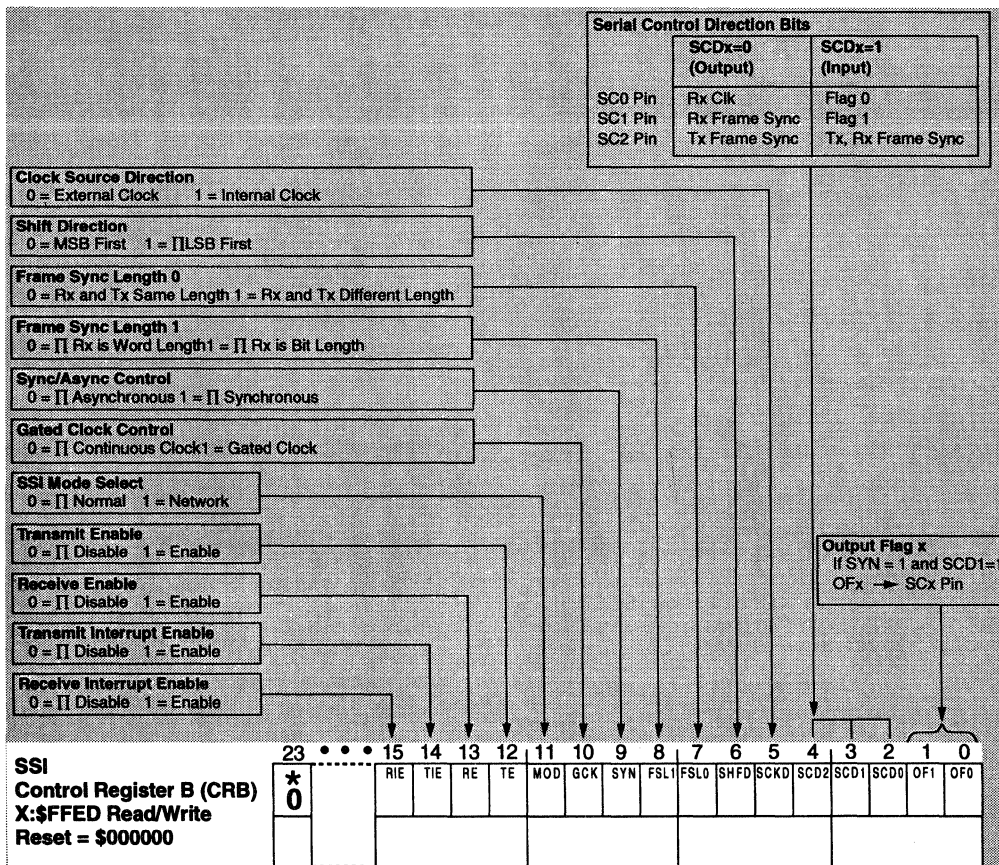


Fig. 5.11 SSI Programming Model—Control and Status Registers



(a) SSI Control Register A (CRA)

* = Reserved, program as zero



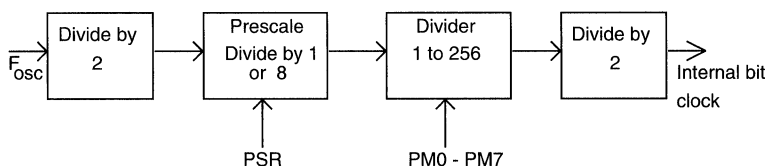


Fig. 5.13 Generation of Internal Bit Clock

clock and an external frame sync signal from the CS4215 audio codec. In both modes, the direction of the serial clock (SCK) and frame sync (SC2) pins are controlled by the CRB control register.

5.3.5 SSI Port Control Register “B” (CRB)

The operation of the SSI port is directed by two 16-bit read/write Control Registers, CRA and CRB. CRB controls the SSI port’s:

- A.** multifunction pins SC2, SC1, and SC0
- B.** serial output flag control bits
- C.** input versus output direction of pins SCK, SC2, SC1, and SC0
- D.** operating modes
- E.** transmitter and receiver enables
- F.** transmitter and receiver interrupt enables

In the DSP System shown in Figs. 5.1 and 5.2, the SSI port’s SC1 and SC0 pins are not used. CRB occupies X-Memory location \$FFED.

In the control mode, the DSP56002 instruction codes shown below set up CRB where:

- A.** CRB bit 4 is set to select the direction of the SC2 pin to be an output (internal frame sync).
- B.** CRB bit 5 is set to select the direction of the SCK pin to be an output (internal clock for both the Transmit and Receive Shift Registers).
- C.** CRB bit 6 is cleared to select the Transmit Shift Register to transmit data MSB first and the Receive Shift Register to receive data MSB first.
- D.** CRB bit 7 is cleared and CRB bit 8 is set to select a one-bit clock period frame sync.
- E.** CRB bit 9 is set to select both the transmitter and the receiver to operate in the synchronous mode and use a common clock (SCK pin) and a common frame sync (SC2 pin).
- F.** CRB bit 10 is cleared to select the “continuous” clock operating mode.
- G.** CRB bit 11 is set to select the “network” operating mode where one word is possibly transferred every DSP56002 time slot.
- H.** CRB bits 12 and 13 are both set to enable the transmitter and receiver, respectively.

- I. CRB bits 14 and 15 are both set to enable the transmit and receive interrupts, respectively.
- J. CRB bits 3–0 are not pertinent to this particular SSI port configuration; nevertheless, they are cleared.

```
CRB    equ    $FFED
      movep   #$FB30,x:CRB    ; RIE,TIE,RE,TE, NTWK, SYN, FSR/RSR->bit
```

In the data mode, the DSP56002 instruction codes shown below set up CRB where:

- A. CRB bit 4 is cleared to select the direction of the SC2 pin to be an input (external frame sync).
- B. CRB bit 5 is cleared to select the direction of the SCK pin to be an input (external clock for both the Transmit and Receive Shift Registers).
- C. CRB bit 6 is cleared to select the Transmit Shift Register to transmit data MSB first and the Receive Shift Register to receive data MSB first.
- D. CRB bit 7 is cleared and CRB bit 8 is set to select a one-bit clock period frame sync.
- E. CRB bit 9 is set to select both the transmitter and the receiver to operate in the synchronous mode and use a common clock (SCK pin) and a common frame sync (SC2 pin).
- F. CRB bit 10 is cleared to select the “continuous” clock operating mode.
- G. CRB bit 11 is set to select the “network” operating mode where one word is possibly transferred every DSP56002 time slot.
- H. CRB bits 12 and 13 are both set to enable the transmitter and receiver respectively.
- I. CRB bits 14 and 15 are both set to enable the transmit and receive interrupts respectively.
- J. CRB bits 3–0 are not pertinent to this particular SSI port configuration; nevertheless, they are cleared.

```
CRB    equ    $FFED
      movep   #$FB00,x:CRB    ; rcv,xmt & int ena,netwk,syn,sclk==i/p,msb 1s
```

Note that the serial clock and frame sync are internal in the data mode and external in the data mode.

5.3.6 SSI Port Receive Registers

The SSI port's Receive Shift Register is a 24-bit register that receives incoming data from the Serial Receive Data (SRD) pin of the SSI port. The Receive Data Register (RX) is a 24-bit register that accepts data in parallel from the Receive Shift Register when the Receive Shift Register becomes full. RX occupies the X-Memory location \$FFEF. The Receive Data Register Full Flag (RDF) of the SSI port's Status Register (Bit 7) is set when the contents of the Receive Shift Register are transferred to RX.

In the data mode, the Receive Shift Register accepts serial input data via the Serial Receive Data (SRD) pin when an external frame sync asserts Serial Control 2 (SC2) pin. The Receive Shift Register is externally clocked via the Serial Clock (SCK) pin. In the control mode, an internal frame sync and an internal clock are used instead of the external ones to accept serial input data via the SRD pin.

5.3.7 SSI Port Transmit Registers

The SSI port's Transmit Shift Register is a 24-bit register that transmits (outputs) serial data via the Serial Transmit Data (STD) pin of the SSI port. The Transmit Data Register (TX) is a 24-bit register that supplies data in parallel to the Transmit Shift Register. Data to be transmitted is written into the TX Register and is automatically transferred to the Transmit Shift Register when a frame sync is asserted. TX occupies X-Memory location \$FFEF. The Transmit Data Register Empty Flag (TDE) of the SSI port's Status Register (Bit 6) is set when the contents of TX are transferred to the Transmit Shift Register.

In the data mode, the Transmit Shift Register transmits serial output data via the Serial Transmit Data (SRD) pin when an external frame sync asserts Serial Control Pin 2 (SC2). The Transmit Shift Register is externally clocked via the Serial Clock (SCK) pin. In the control mode, an internal frame sync and an internal clock are used to transmit serial output data via the SRD pin.

5.3.8 SSI Port Status Register

The SSI port's Status Register (SSISR) shown in Fig. 5.11 is an 8-bit read-only register used by the DSP56002 processor to interrogate the status and serial input flags of the SSI port. The Status Register occupies X-Memory location \$FFEE.

The SSISR Receive Data Register Full (RDF) bit (Bit 7) is set when the contents of the Receive Shift Register are transferred to the Receive Data Register. RDF is cleared when the DSP56002 processor reads the Receive Data Register, or it is cleared by hardware, software, SSI individual, or STOP reset.

The SSISR Transmit Data Register Empty (TDE) flag bit (Bit 6) is set when the contents of the Transmit Data Register are transferred to the Transmit Shift Register; it also is set for a disabled time slot period in network mode. TDE is cleared when the DSP56002 processor writes new data to TX or when the DSP56002 processor writes to the Time Slot Register (TSR) to disable transmission of the next time slot.

The SSISR Receiver Overrun Error (ROE) flag bit (Bit 5) is set when the Serial Receive Shift Register is filled and ready to transfer to the Receiver Data Register (RX) and RX is already full (i.e., RDF = 1). The ROE is cleared with hardware, software, SSI individual, and STOP reset. ROE is also cleared by reading the SSISR with ROE set, followed by reading the RX.

The SSISR Transmitter Underrun Error (TUE) flag bit (Bit 4) is set when the Serial Transmit Shift Register is empty (no new data to be transmitted) and a DSP56002 transmit time slot occurs. When a transmit underrun error occurs, the previous data (which is still present in the TX) will be retransmitted. The TUE is cleared

with hardware, software, SSI individual, and STOP reset. TUE is also cleared by reading the SSISR with TUE set, followed by writing TX or Time Slot Register (TSR).

The SSISR Serial Input Flags 0 and 1 (IF0 and IF1) bits (Bits 0 and 1) are not used in this DSP configuration.

When the SSISR Transmit Frame Sync (TFS) flag bit is set (Bit 2), it indicates that a transmit frame sync occurred in the current DSP56002 time slot. TFS is set at the start of the first time slot in the frame and cleared during all other DSP56002 time slots. TFS is useful in network mode to identify the start of the frame.

In the DSP system, the following DSP56002 instruction codes detect the start of the frame in the network to establish the necessary synchronization needed before the use of short interrupts to transfer data between the DSP56002 processor and the two A/D and two D/A converters.

```

do      #$100,gothru
jclr   #2,x:SSISR,*           ; wait until tx frame bit==1
jset   #2,x:SSISR,*           ; wait until tx frame bit==0
nop
gothru

```

When the SSISR Receive Frame Sync (RFS) flag bit is set (Bit 3), it indicates that a receive frame sync occurred during reception of the word in the serial receive data register. This indicates that the data word is from the first DSP56002 time slot in the frame. RFS is useful in the network mode to identify the start of the frame.

In the DSP system, the following DSP56002 instruction codes are part of the initialization codes needed to initialize the CS4215 audio codec before going from the control mode to the data mode.¹

```

movep   #$01E8,x:PCC           ; Turn on ssi port
;
; CLB == 0
;
jclr    #3,x:SSISR,*           ; wait until rx frame bit==1
jset    #3,x:SSISR,*           ; wait until rx frame bit==0
jclr    #3,x:SSISR,*           ; wait until rx frame bit==1
jset    #18,x:RX_BUFF_BASE,*   ; loop until CLB set
;
; CLB == 1
;
bset    #18,x:TX_BUFF_BASE     ;set CLB
do      #4,_init_loopB
jclr    #2,x:SSISR,*           ; wait until tx frame bit==1
jset    #2,x:SSISR,*           ; wait until tx frame bit==0
_init_loopB
movep   #0,x:PCC               ; disable, reset SSI

```

1. CS 4215 Multimedia Audio Codec Data Sheet, Revision E, Crystal Semiconductor Corporation, October 1994.

The following DSP56002 instruction codes and macros are used to route the digitized form of the left and right analog signals received from the two A/D converters to the left and right D/A converters.

```

data_loop
    wait_receive
    wait_word

    get_left
    get_right

    move x0,a
    put_left
    move x1,b
    put_right

    jmp data_loop

wait_receive macro ; this macro wait to receive data
    jclr    #3,x:SSISR,*           ; wait until rx frame bit==1
    jset    #3,x:SSISR,*           ; wait until rx frame bit==0
endm

wait_send macro
    jclr    #2,x:SSISR,*           ; wait until tx frame bit==1
    jset    #2,x:SSISR,*           ; wait until tx frame bit==0
endm

get_left macro ; this macro get the left channel to x0
    move    x:RX_BUFF_BASE,x0
endm

put_left macro ; this macro put the left channel from a
    move    a,x:TX_BUFF_BASE
endm

get_right macro ; this macro get the right channel to x1
    move    x:RX_BUFF_BASE+1,x1
endm

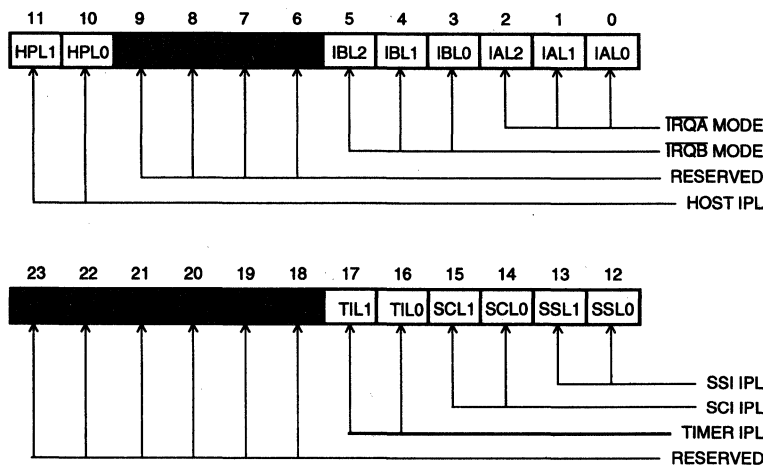
put_right macro ; this macro put the right channel from b
    move    b,x:TX_BUFF_BASE+1
endm

wait_word macro
    jclr    #1,r7,*
endm

```

5.4 DSP56002 INTERRUPT PRIORITY REGISTER

Interrupt priority levels for each on-chip peripheral device and for each external interrupt source can be programmed under software control by writing to the Interrupt Priority Register (IPR). Level 3 interrupts are nonmaskable, and interrupts of level 0–2 are maskable. The DSP56002 IPR configuration is shown in Fig. 5.14. The following DSP56002 instruction codes program the interrupt priority level of the IPR as level 3



Reserved, read as zero and should be written with zero for future compatibility.

Fig. 5.14 DSP56002 Interrupt Priority Register

(nonmaskable). The interrupt mask bits (Bits 8 and 9) in the Mode Register (MR) are cleared to permit exception levels 0, 1, 2, and 3.

```
movep    #$3000,x:IPR      ; set interrupt priority level
andi     #$FC,mr           ; enable interrupts
```

The starting addresses of interrupt vectors in the DSP56002 are defined in Table 5.3, while the relative priorities of exceptions within the same IPL are defined in Table 5.4. The SSI Receive Data interrupt source is at the interrupt starting address P:\$C. The SSI Receive Data with Exception Status interrupt source is at the interrupt starting address P:\$E. The SSI Transmit Data interrupt source is at the interrupt starting address P:\$10. The SSI Transmit Data with Exception Status interrupt source is at the interrupt starting address P:\$12. The following DSP56002 instruction codes and subroutines generate the two SSI interrupt sources used to move data stored in the transmit buffer to the SSI Receive\Transmit Data Register SSIDR (memory location X:\$FFEF). Also, the data received in the SSI Receive\Transmit Data Register SSIDR is moved to the receive buffer.

```
org      p:$C
jsr      ssi_rx_isr          ;SSI RX
jsr      ssi_rx_isr          ;SSI RX w/Exception
jsr      ssi_tx_isr          ;SSI TX
jsr      ssi_tx_isr          ;SSI TX w/Exception

;***** SSI TRANSMIT ISR *****
ssi_tx_isr
    jclr    #2,x:SSISR,next_tx ; If not frame sync, jump to transmit data.
```

```

        move    #TX_BUFF_BASE+1,r0        ; If frame sync, reset pointer.
        nop
next_tx
        movep   x:(r0)+,x:SSIDR           ; SSI transfer data register.
        rti

;***** SSI receive ISR*****
ssi_rx_isr

```

Table 5.3 Interrupt Vectors

Interrupt Starting Address	IPL	Interrupt Source
P:\$0000	3	Hardware RESET
P:\$0002	3	Stack Error
P:\$0004	3	Trace
P:\$0006	3	SWI
P:\$0008	0 - 2	IRQA
P:\$000A	0 - 2	IRQB
P:\$000C	0 - 2	SSI Receive Data
P:\$000E	0 - 2	SSI Receive Data With Exception Status
P:\$0010	0 - 2	SSI Transmit Data
P:\$0012	0 - 2	SSI Transmit Data with Exception Status
P:\$0014	0 - 2	SCI Receive Data
P:\$0016	0 - 2	SCI Receive Data with Exception Status
P:\$0018	0 - 2	SCI Transmit Data
P:\$001A	0 - 2	SCI Idle Line
P:\$001C	0 - 2	SCI Timer
P:\$001E	3	NMI
P:\$0020	0 - 2	Host Receive Data
P:\$0022	0 - 2	Host Transmit Data
P:\$0024	0 - 2	Host Command (Default)
P:\$0026	0 - 2	Available for Host Command
P:\$003A	0 - 2	Available for Host Command
P:\$003C	0 - 2	Timer
P:\$003E	3	Illegal Instruction
P:\$0040	0 - 2	Available for Host Command
P:\$007E	0 - 2	Available for Host Command

```

jclr    #3,x:SSISR,next_rx    ; If not frame sync, jump to receive data.
move    #RX_BUFF_BASE,r7     ; If frame sync, reset base pointer.
nop
next_rx
movep    x:SSIDR,x:(r7)+      ; Read out received data to buffer.
rti

```

Table 5.4 Exception Priorities within an Interrupt Priority Level

Priority	Exception
Level 3 (Nonmaskable)	
Highest	Hardware RESET
	Illegal Instruction
	NMI
	Stack Error
	Trace
Lowest	SWI
Levels 0, 1, 2 (Maskable)	
Highest	IRQA (External Interrupt)
	IRQB (External Interrupt)
	Host Command Interrupt
	Host Receive Data Interrupt
	Host Transmit Data Interrupt
	SSI RX Data with Exception Interrupt
	SSI RX Data Interrupt
	SSI TX Data with Exception Interrupt
	SSI TX Data Interrupt
	SCI RX Data with Exception Interrupt
	SCI RX Data Interrupt
	SCI TX Data with Exception Interrupt
	SCI TX Data Interrupt
	SCI Idle Line Interrupt
	SCI Timer Interrupt
Lowest	Timer Interrupt

Note that Modulo Addressing is used to address the transmit and receive buffers. Modulo addressing is described in detail in chapter 6.

5.5 INITIALIZATION MACRO

The **init.asm** assembler program shown in Fig. 5.15 is used to initialize the DSP system shown in Figs. 5.1 and 5.2.

Fig. 5.15 Init.asm Assembler Program

```

;*****
;
;   INIT.ASM  ASSEMBLER PROGRAM
;
;*****
;

init_system macro

DEBUG          equ          $000200
NO_PREAMP      equ          $100000
SAMP_RATE_48   equ          $003000
STEREO         equ          $000400
DATA_16        equ          $200000
IMMED_3STATE   equ          $800000
XTAL2_SELECT   equ          $200000
BITS_64        equ          $000000
CODEC_MASTER   equ          $020000

CTRL_WD_12     equ          NO_PREAMP+SAMP_RATE_48+STEREO+DATA_16 ;CLB=0
CTRL_WD_34     equ          IMMED_3STATE+XTAL2_SELECT+BITS_64+CODEC_MASTER
CTRL_WD_56     equ          $000000
CTRL_WD_78     equ          $000000

HEADPHONE_EN   equ          $800000
LINEOUT_EN     equ          $400000
LEFT_ATTN      equ          $010000 ;63*LEFT_ATTN   = -94.5 dB, 1.5 dB steps
RIGHT_ATTN     equ          $000100 ;63*RIGHT_ATTN  = -94.5 dB, 1.5 dB steps
MIC_IN_SELECT  equ          $100000
MONITOR_ATTN   equ          $001000 ;15*MONITOR_ATTN = mute,      6 dB steps
OUTPUT_SET     equ          HEADPHONE_EN+LINEOUT_EN+(LEFT_ATTN*4)
INPUT_SET      equ          MIC_IN_SELECT+(15*MONITOR_ATTN)+(RIGHT_ATTN*4)

;   DSP56002 on-chip peripheral addresses

PCD            equ          $FFE5
PCDDR          equ          $FFE3
PCC            equ          $FFE1
CRA            equ          $FFEC
CRB            equ          $FFED
SSIDR          equ          $FFEF
IPR            equ          $FFFF
BCR            equ          $FFFE
SSISR          equ          $FFEE
PLL            equ          $FFFD

org            x:0

```

```

RX_BUFF_BASE    equ      *
RX_data_1_2     ds       1      ;data time slot 1/2 for RX ISR
RX_data_3_4     ds       1      ;data time slot 3/4 for RX ISR
RX_data_5_6     ds       1      ;data time slot 5/6 for RX ISR
RX_data_7_8     ds       1      ;data time slot 7/8 for RX ISR

TX_BUFF_BASE    equ      *
TX_data_1_2     ds       1      ;data time slot 1/2 for TX ISR
TX_data_3_4     ds       1      ;data time slot 3/4 for TX ISR
TX_data_5_6     ds       1      ;data time slot 5/6 for TX ISR
TX_data_7_8     ds       1      ;data time slot 7/8 for TX ISR

SSI_FLAG        ds       1      ;bit flags used for word index
RX_PTR          ds       1      ; Pointer for rx buffer
TX_PTR          ds       1      ; Pointer for tx buffer

init_codec      equ      $40

                org      p:0
                jmp      START

                org      p:$C

                jsr      ssi_rx_isr      ;SSI RX
                jsr      ssi_rx_isr      ;SSI RX
                jsr      ssi_tx_isr      ;SSI TX
                jsr      ssi_tx_isr      ;SSI TX

;-----
;
;      Main init macro
;
;-----
                org      p:init_codec

;*****
; initialize the CRYSTAL CS4215 codec
;*****
; headphones and line out, and set up for no gain or attenuation, and no
; monitor feedback.
;*****
;
;      initialize ssi -- fsync and sclk == outputs
;
                movep    #$0000,x:PCC      ; turn off ssi port
                movep    #$4303,x:CRA      ; 40MHz/16 = 2.5MHz SCLK, WL=16 bits, 4W/F
                movep    #$FB30,x:CRB      ; RIE,TIE,RE,TE, NTWK, SYN, FSR/RSR-bit
                movep    #$14,x:PCDDR      ; setup pc2 and pc4 as outputs
                movep    #$0,x:PCD        ; D/C~ and RESET~ = 0 == control mode

                                ;----reset delay for codec ----
                do        #500,delay_loop
                rep        #2000          ; 100 us delay
                nop

delay_loop
                nop

                bset     #4,x:PCD          ; RESET~ = 1
                movep    #$3000,x:IPR      ; set interrupt priority level
                andi     #$FC,mr          ; enable interrupts

```

```

;--- set up buffer with control mode data

    move    #CTRL_WD_12,x0
    move    x0,x:TX_BUFF_BASE
    move    #CTRL_WD_34,x0
    move    x0,x:TX_BUFF_BASE+1
    move    #CTRL_WD_56,x0
    move    x0,x:TX_BUFF_BASE+2
    move    #CTRL_WD_78,x0
    move    x0,x:TX_BUFF_BASE+3

    movep   #$01E8,x:PCC      ; Turn on ssi port

;
; CLB == 0
;
    jclr    #3,x:SSISR,*      ; wait until rx frame bit==1
    jset    #3,x:SSISR,*      ; wait until rx frame bit==0
    jclr    #3,x:SSISR,*      ; wait until rx frame bit==1
    jset    #18,x:RX_BUFF_BASE,* ; loop until CLB set

;
; CLB == 1
;
    bset    #18,x:TX_BUFF_BASE ;set CLB
    do      #4,init_loopB
    jclr    #2,x:SSISR,*      ; wait until tx frame bit==1
    jset    #2,x:SSISR,*      ; wait until tx frame bit==0
init_loopB
    movep   #0,x:PCC          ;disable, reset SSI

;*****
;    now CLB should be 1 -- re-program fsync and sclk direction (i/p) --

    movep   #$4303,x:CRA      ; 16bits,4 word/frame, /2/4/2=2.5 MHz
    movep   #$FB00,x:CRB      ; rcv,xmt & int ena,netwk,syn,sclk==i/p,msb 1st
    movep   #$14,x:PCD        ; D/C~ pin = 1 == data mode
    movep   #$01E8,x:PCC      ; turn on ssi port

    rts

; ***** SSI TRANSMIT ISR *****

ssi_tx_isr
    jclr    #2,x:SSISR,next_tx ; If not frame sync, jump to transmit data.
    move    #TX_BUFF_BASE+1,r0 ; If frame sync, reset pointer.
    nop
next_tx
    movep   x:(r0)+,x:SSIDR     ; SSI transfer data register.
    rti

;***** SSI RECEIVE ISR*****

ssi_rx_isr
    jclr    #3,x:SSISR,next_rx ; If not fr. sync, jump to receive data.
    move    #RX_BUFF_BASE,r7   ; If frame sync, reset base pointer.
    nop
next_rx
    movep   x:SSIDR,x:(r7)+     ; Read out received data to buffer.
    rti

;***** init_system macro *****

```


START

```

movep    #$261009,x:PLL          ;set PLL for MPY of 10x
movep    #$0000,x:BCR            ; number of wait states
ori       #3,mr                  ;disable interrupts
movec     #0,sp
move      #0,omr                  ; single chip mode

move      #3,m0                  ; Modulus 4 buffer.
move      #3,m7                  ; Modulo 4 buffer.
move      #RX_BUFF_BASE,x0
move      x0,x:RX_PTR            ; Initialize the rx pointer
move      x:RX_PTR,r7            ; Load the pointer to the rx buffer.
move      #TX_BUFF_BASE+1,x0
move      x0,x:TX_PTR            ; Initialize the tx pointer
move      x:TX_PTR,r0            ; Load the pointer to the rx buffer.
jsr       init_codec

```

; init the output set in the beginning rather than each time

```

move      #OUTPUT_SET,y0          ;headphones, line out, mute spkr, no attn
move      y0,x:TX_BUFF_BASE+2
move      #INPUT_SET,y0           ;no input gain, monitor mute
move      y0,x:TX_BUFF_BASE+3

```

; wait for sync to be established and then switch to shorter interrupts

```

do        #$100,gothru
jclr      #2,x:SSISR,*            ; wait until tx frame bit==1
jset      #2,x:SSISR,*            ; wait until tx frame bit==0
nop

```

gothru

nop

; set up shorter interrupts to next_rx for receiving and next_tx for trans.

```

move      #next_rx,a1
move      a1,p:$d
move      #next_tx,a1
move      a1,p:$11
nop
endm

```

wait_receive macro ; this macro wait to receive data

```

jclr      #3,x:SSISR,*            ; wait until rx frame bit==1
jset      #3,x:SSISR,*            ; wait until rx frame bit==0
endm

```

wait_send macro

```

jclr      #2,x:SSISR,*            ; wait until tx frame bit==1
jset      #2,x:SSISR,*            ; wait until tx frame bit==0
endm

```

get_left macro ; this macro get the left channel to x0

```

move      x:RX_BUFF_BASE,x0
endm

```

put_left macro ; this macro put the left channel from a

```

move      a,x:TX_BUFF_BASE
endm

```

get_right macro ; this macro get the right channel to x1

```

move      x:RX_BUFF_BASE+1,x1

```

```

        endm

put_right macro          ; this macro put the right channel from b
        move     b,x:TX_BUFF_BASE+1
        endm

wait_word macro
        jclr    #1,r7,*
        endm

```

5.6 EXERCISING THE DSP SYSTEM

To exercise the DSP System, explained in section 5.1, two analog input signals are provided from two function generators to the left and right analog inputs of the EVM. The analog input signals are monitored using a dual trace oscilloscope. The analog output signals from the EVM are monitored using a second dual trace oscilloscope.

The DSP56002 processor located on the EVM executes the absolute object file **dsp.cld** to route the digitized form of the left and right analog signals. The file **dsp.cld** is the assembled result of the assembler program shown in Fig. 5.16 and developed in the previous sections.

1. Power down the PC and function generators.
2. Connect the two function generators' outputs to the EVM's "in" input (two A/D inputs).
3. Connect the two function generators' outputs to the first oscilloscope's two channel inputs.
4. Connect the EVM's "HDPHNE" output (two D/A outputs) to the second oscilloscope's two channel inputs and to a headset.
5. Power on the two function generators.
6. Set each function generator to output a 1-volt (peak-to-peak), 5 KHz sine wave.
7. Power on and boot the PC.
8. Insert the "DSP56002 Demonstration Software #1" diskette into Drive "A."
9. Invoke the Debug-EVM program.
10. Type change omr 0<return>
 load a:dsp.cld<return>
 to load the assembled version of the assembler program shown in Fig. 5.16.
11. Type go<return> to begin execution of the program.
12. Observe the input and output signals on the two oscilloscopes. Note that the output signals are good reconstruction of the input signals.
13. Vary each function generator's output frequency (f) from 0.5 to 20 KHz.
14. Observe the input and output signals on the two oscilloscopes.

Note that the output signals are good reconstruction of the input signals at frequencies from 0.5 KHz to 20 KHz. This should be the case since the 48 KHz

sampling rate selected for the two A/D converters and the D/A converters satisfies the Nyquist Sampling Theorem for input frequencies below 24 KHz, e.g., $2 \times 24 \text{ KHz} = 48 \text{ KHz}$, the Nyquist sampling rate for input frequencies up to 24 KHz. It is assumed here that the 20 KHz frequency is lower than the cutoff frequency of any existing reconstruction filter.

15. Set each function generator to output a 1-volt amplitude, 5 KHz square wave.
16. Observe the input and output signals on the two oscilloscopes.
17. Vary the frequency (f) of each square wave input signal from 0.5 To 20 KHz.
18. Observe the input and output signals on the two oscilloscopes.

Recall that a square wave signal can be decomposed into an infinite number of odd harmonic sine wave signals. It follows then that the output signal is reconstructed from odd harmonic sine wave signals having frequencies (nf) less than one-half the 48 KHz sampling frequency and less than the cutoff frequency of any existing reconstruction filter. As the frequency of the input square wave signal approaches 20 KHz, fewer and fewer odd harmonic components are available to contribute to the reconstruction of the signal. As a result, the quality of the signal reconstruction suffers.

19. Connect a stereo audio signal to the EVM's "in" input.
20. Use the headset to listen to the stereo output from the EVM's "HDPHNE."
21. Type force b<return> to halt execution of the program.
22. Type quit<return> if you want to exit the Debug-EVM Software program.

Fig. 5.16 DSP.ASM Assembler Program for Reading two A/Ds and Writing two D/As

```
;*****
;
; program to pass input to output without processing
;
;*****

    include 'init.asm'

    init_system
    nop

data_loop

    wait_receive
    wait_word

    get_left
    get_right

    move x0,a
    put_left
    move x1,b
    put_right

    jmp data_loop
```

Designing FIR Filters and Implementing Them on the DSP56002 Processor

In this chapter, Finite Impulse Response (FIR) filters are designed and implemented on the DSP56002 processor. The FIR filter has three distinct properties:

1. it is always stable;
2. it is always realizable; and
3. it can always be designed to have a linear phase response.

The transfer function and impulse response of a FIR filter are defined in this chapter, followed by a description of how the ideal impulse response of an ideal FIR filter is obtained from its frequency response. Since the ideal impulse response sequence of a FIR filter is infinitely long, an appropriate “window” function is designed and used to create a practical, finite length impulse response sequence.

In the design process for a practical FIR filter, its desired frequency domain specifications are used with an appropriate window function to determine its impulse response $h(i)$. The DSP56002 Demonstration Software program, and the Digital Filter Design And Analysis System software program (QEDesign 1000 for Windows) by Momentum Data Systems Inc., are used to demonstrate this design process. The DSP56002 instruction codes that implement the designed FIR filter on the DSP56002 processor are then described.

Low-pass, high-pass, band-pass, and band-stop FIR filters are then designed and implemented on the DSP system developed in chapter 5. The DSP56002 instruction codes that implement these four FIR filter types are included on the DSP56002 Demonstration Software diskette.

6.1 FIR FILTER FREQUENCY RESPONSE

The relationship between a FIR filter's digital input sequence $x(n)$ and digital output sequence $y(n)$ can be written as:

$$y(n) = \sum_{i=0}^{N-1} b_i x(n-i) \quad (6.1)$$

where b_i are filter coefficients and N is the number of those coefficients. The filter coefficients b_i provide the characteristics of the filter through which the input sequence $x(n)$ is passed to create the desired output sequence $y(n)$. By both definition and design, the output sequence $y(n)$ of a FIR filter (also known as a nonrecursive filter) is a function of its current and past inputs only.

The z-Transform transfer function $H(z)$ corresponding to Equation 6.1 can be expressed as:

$$\begin{aligned} H(z) &= \frac{Y(z)}{X(z)} = \sum_{i=0}^{N-1} b_i z^{-i} \\ &= \sum_{i=0}^{N-1} h(i) z^{-i} \end{aligned} \quad (6.2)$$

where $h(i)$, the impulse response of the FIR filter, is composed of b_i , the coefficients of the FIR filter.

To determine the frequency response of the FIR filter, the z-Transform variable (z) is evaluated on the unit circle as:

$$z = e^{j2\pi f} \quad (6.3)$$

where f = normalized frequency, i.e., the actual frequency divided by the sampling frequency.

By substituting Equation 6.3 into Equation 6.2, the frequency response of the FIR filter can be written as:

$$H(e^{j2\pi f}) = \sum_{i=0}^{N-1} h(i) e^{-j2\pi fi} \quad (6.4)$$

6.2 RELATING A PRACTICAL FIR FILTER TO AN IDEAL FIR FILTER

In the design process for a practical FIR filter, its desired frequency domain specifications are used with an appropriate window function to determine its impulse response $h(i)$. The coefficients $h(i)$ (or b_i) of the transfer function $H(z)$ in Equation 6.2 are designed so that the frequency response in Equation 6.4 meets certain frequency

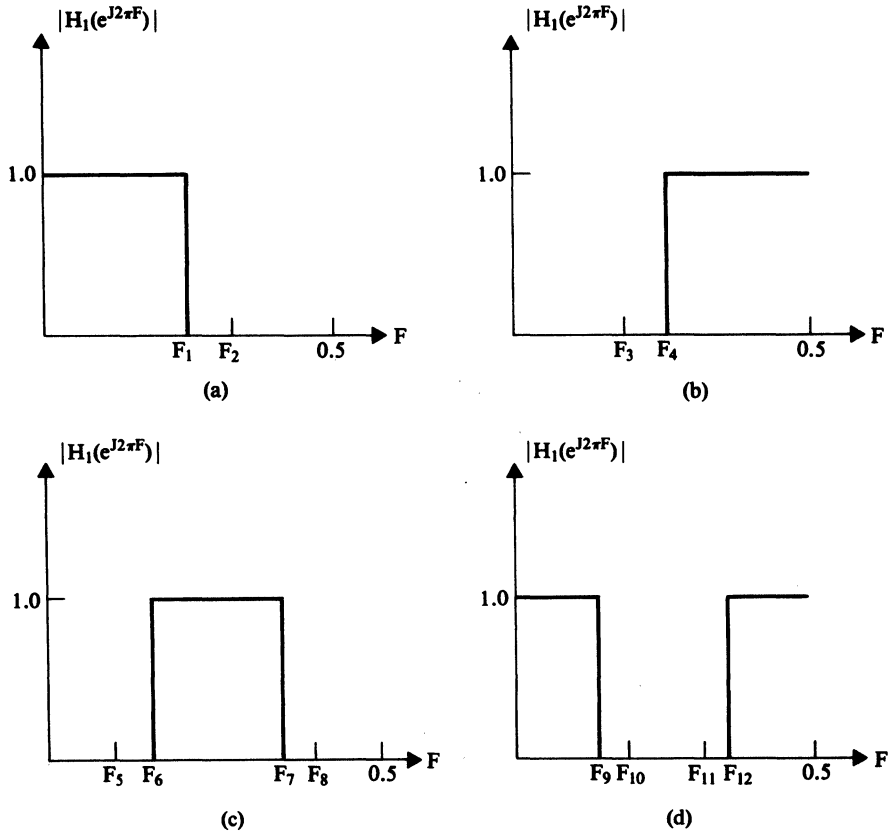


Fig. 6.1 Ideal Filters: a) Low-pass; b) High-pass; c) Band-pass; d) Band-stop

domain specifications. Once the impulse response is obtained, its b_i components are used to implement Equation 6.1 on the DSP56002 processor.

The first step in the design process is to obtain the frequency response of an ideal FIR filter. By definition, an ideal FIR filter has a magnitude of one in the passbands and zero elsewhere. The magnitude of the frequency response of an ideal low-pass, high-pass, band-pass, and band-stop filter is shown in Fig. 6.1.

The frequency response of the ideal FIR filter can be written as:

$$H_1(e^{j2\pi f}) = \sum_{i=-\infty}^{\infty} h_1(i)e^{-j2\pi fi} \quad (6.5)$$

where $h_1(i)$, the ideal impulse response, can be calculated as:

$$h_1(i) = \int_{-0.5}^{0.5} H_1(e^{j2\pi f})e^{j2\pi fi} df \quad (6.6)$$

and 0.5 is half the normalized sampling frequency.

A practical FIR filter can be implemented only if its impulse response is of finite length. Since the ideal impulse response $h_1(i)$ is an infinitely long sequence, it must be limited to a finite length. To do so, $h_1(i)$ is multiplied by an N -point window. The N -point impulse response $h_2(i)$ is then given by:

$$\begin{aligned} h_2(i) &= h_1(i)w(i) & -(N-1)/2 \leq i \leq (N-1)/2 \\ &= 0 & \text{otherwise} \end{aligned} \quad (6.7)$$

where N is an odd-valued integer. Some of the commonly used window types are shown in Fig. 6.2.

Since a causal filter has zero impulse response for i less than zero, an FIR filter can be made to satisfy this condition by sufficiently delaying its impulse response (if necessary). That is, $h_2(i)$ can be delayed by $(N-1)/2$ so that the desired impulse response $h(i)$ of a causal FIR filter can be obtained as:

$$h(i) = h_2[i - ((N-1)/2)], \quad \text{where } i = 0, 1, 2, \dots, N-1 \quad (6.8)$$

A causal FIR filter can be made to have a linear phase response by choosing $h(i)$ to be equal to $h(N-1-i)$ for $i = 0, \dots, (N-1)/2$. With this choice for $h(i)$, Equation 6.4 can be rewritten as:

$$H(e^{j2\pi f}) = \sum_{i=0}^{(N-3)/2} h(i)e^{-j2\pi fi} + h(0.5(N-1))e^{-j2\pi f(N-1)/2} + \sum_{i=(N+1)/2}^{N-1} h(i)e^{-j2\pi fi} \quad (6.9)$$

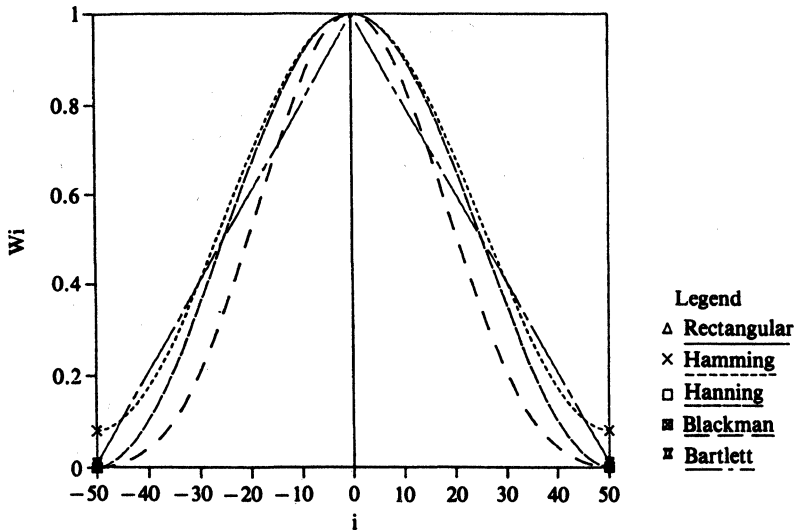


Fig. 6.2 Commonly Used Windows

By letting $N-1-i=i$, the last summation in Equation 6.9 can be expressed as:

$$\begin{aligned}
 \sum_{i=(N+1)/2}^{N-1} h(i)e^{-j2\pi fi} &= \sum_{i=(N+1)/2}^{N-1} h(N-1-i)e^{-j2\pi fi} \\
 &= \sum_{i=0}^{(N-3)/2} h(i)e^{-j2\pi f(N-1-i)}
 \end{aligned} \tag{6.10}$$

From Equations 6.9 and 6.10:

$$\begin{aligned}
 H(e^{j2\pi f}) &= e^{-j2\pi f(N-1)} \left[h(0.5(N-1)) + \sum_{i=0}^{(N-3)/2} h(i)(e^{j2\pi f(N-1-2i)} + e^{-j2\pi f(N-1-2i)}) \right] \\
 &= e^{-j2\pi f(N-1)} \left[h(0.5(N-1)) + \sum_{i=0}^{(N-3)/2} 2h(i)\cos(\pi f(N-1-2i)) \right]
 \end{aligned} \tag{6.11}$$

For real $h(i)$, the term between the two brackets in Equation 6.11 is a real function of f . If this term is positive, then the phase of $H(e^{j2\pi f})$ is equal to $-\pi f(N-1)$. If this term is negative, then the phase of $H(e^{j2\pi f})$ is equal to $\pi - \pi f(N-1)$. In either case, the phase of $H(e^{j2\pi f})$ is a linear function of f .

6.3 SPECIFYING A PRACTICAL FIR FILTER

The frequency domain specifications for practical FIR filters include the passband and stopband attenuations, and the passband and stopband cutoff frequencies. Figure 6.3 shows the frequency domain specifications for low-pass, high-pass, band-pass, and band-stop FIR filters, where:

- α = Passband attenuation in dB
- β = Stopband attenuation in dB
- f_1 = Normalized passband cutoff frequency of the low-pass filter
- f_2 = Normalized stopband cutoff frequency of the low-pass filter
- f_3 = Normalized stopband cutoff frequency of the high-pass filter
- f_4 = Normalized passband cutoff frequency of the high-pass filter
- f_5 = Normalized lower stopband cutoff frequency of the band-pass filter
- f_6 = Normalized lower passband cutoff frequency of the band-pass filter
- f_7 = Normalized upper passband cutoff frequency of the band-pass filter
- f_8 = Normalized upper stopband cutoff frequency of the band-pass filter
- f_9 = Normalized lower passband cutoff frequency of the band-stop filter
- f_{10} = Normalized lower stopband cutoff frequency of the band-stop filter

f_{11} = Normalized upper stopband cutoff frequency of the band-stop filter

f_{12} = Normalized upper passband cutoff frequency of the band-stop filter

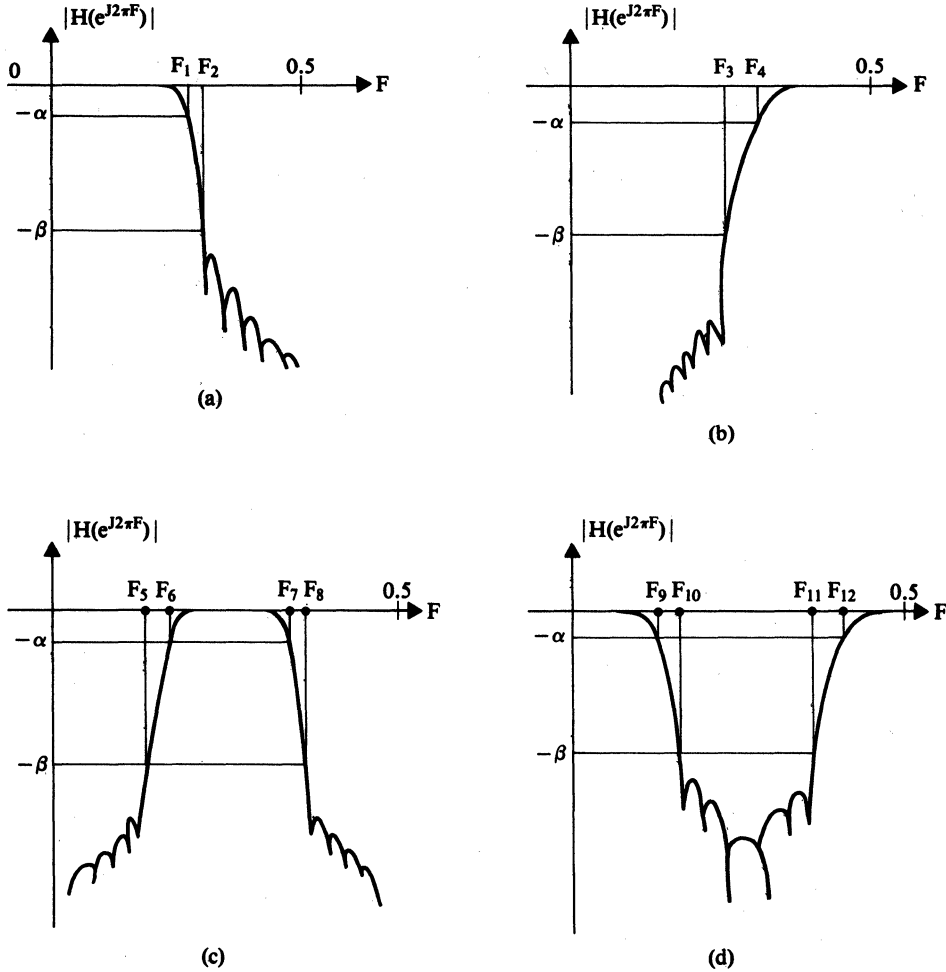


Fig. 6.3 Frequency Domain Specifications: a) Low-pass; b) High-pass; c) Band-pass; d) Band-stop

6.4 USING THE WINDOW METHOD TO DESIGN A PRACTICAL FIR FILTER

There are three well-known methods for designing practical, linear phase response, FIR filters:

- A. the frequency sampling method
- B. the “minimax” method
- C. the window method

The window method is used in this book. The procedure is:

1. Compute the ideal impulse response $h_1(i)$.

By using Equation 6.6, the ideal impulse response $h_1(i)$ of the low-pass, high-pass, band-pass, and band-stop FIR filters can be calculated as:

$$\text{Low-pass: } h_1(i) = \frac{\sin 2\pi f_1 i}{\pi i} \quad (6.12)$$

$$\text{High-pass: } h_1(i) = -\frac{\sin 2\pi f_4 i}{\pi i} \quad (6.13)$$

$$\text{Band-pass: } h_1(i) = \frac{\sin 2\pi f_7 i}{\pi i} - \frac{\sin 2\pi f_6 i}{\pi i} \quad (6.14)$$

$$\text{Band-stop: } h_1(i) = \frac{\sin 2\pi f_9 i}{\pi i} - \frac{\sin 2\pi f_{12} i}{\pi i} \quad (6.15)$$

2. Select the window type.

To achieve a desired level of stopband attenuation, an appropriate window type must be selected from Table 6.1. For example, if the desired stopband attenuation is 50 dB, then, from Table 6.1, either the Hamming, Blackman, or Kaiser window is applicable.

3. Select the window length “N” that corresponds with the selected window type.

Table 6.1 Window Type Selection Table

Window	Stopband Attenuation in dB	K
Rectangular	21	2.00
Bartlett	25	4.00
Hanning	44	4.00
Hamming	53	4.00
Blackman	74	6.00
Kaiser ($\gamma=2.12$)	30	1.54
Kaiser ($\gamma=4.54$)	50	2.93
Kaiser ($\gamma=7.76$)	70	4.32
Kaiser ($\gamma=8.96$)	90	5.71

The following relationship can be used to approximate the necessary length N of the window:

$$N \geq K/(\Delta f) \quad (6.16)$$

where N is the least odd integer that satisfies Equation 6.16, K is a constant selected from Table 6.1, and

Δf = the transition band expressed in Hz.
 = $f_2 - f_1$ for low-pass filter
 = $f_4 - f_3$ for high-pass filter
 = $\min [(f_6 - f_5), (f_8 - f_7)]$ for band-pass filter
 = $\min [(f_{10} - f_9), (f_{12} - f_{11})]$ for band-stop filter

4. From Table 6.2, select the window function $w(i)$ that corresponds with the selected window type.

Fig. 6.4 FORTRAN Subroutine "BESSEL" to Calculate the Modified Bessel Function of Order Zero

```

C*****
C*
C*  SUBROUTINE BESSEL
C*
C*
C*****
C
C      I0(X) = XBESS
C
C
C
C      SUBROUTINE BESSEL (X,XBESS)
C      INTEGER K
C      DOUBLE PRECISION IF
C      S=1.0
C      K=1
C      IF=1
C      E1=10000.0
C      E2=E1
77  IF=IF*K
C      E1=E2
C      E2=((X/2.0)**K/IF)**2
C      S=S+E2
C      ERR1=E2
C      ERR2=E2/E1
C      IF (ERR1.LT.0.00001) GOTO 88
C      IF (ERR2.LT.0.00001) GOTO 88
C      IF (K.GT.100) GOTO 99
C      K=K+1
C      GOTO 77
88  XBESS=S
99  RETURN
END

```

Table 6.2 Window Functions

Window Type	Window Function $w(i)$ $ i \leq (N-1)/2$
Rectangular	1
Bartlett	$1 - 2(i /N)$
Hanning	$0.5 + 0.5 \cos(2\pi i/N)$
Hamming	$0.54 + 0.46 \cos(2\pi i/N)$
Blackman	$0.42 + 0.5 \cos(2\pi i/N) + 0.08 \cos(4\pi i/N)$
Kaiser*	$I_0[\gamma(1-(2i/(N-1))^2)^{0.5}] / I_0(\gamma)$

* I_0 is the modified Bessel function of order zero. The FORTRAN subroutine "BESSEL" shown in Fig. 6.4 can be used to calculate the modified Bessel function of order zero.

- Using equation 6.7, calculate $h_2(i)$.

$h_2(i)$ is calculated by multiplying the appropriate $h_1(i)$ selected from Equations 6.12 through 6.15 by the appropriate $w(i)$ selected from Table 6.2.

- Using equation 6.8, calculate $h(i)$.

Equation 6.4 can be used to compute the magnitude of the frequency response. The frequency response can be computed using the fast Fourier transform presented in chapter 8. The designed FIR filter can then be tested to verify that its desired frequency domain specifications are met.

Note that the value of N obtained from Equation 6.16 is higher than that which is necessary to meet the desired frequency domain specifications of the filter that is being designed. To make an adjustment, N can be decremented (incremented) by an even integer value to reduce (increase) the "order" of the FIR filter. This can be repeated until a filter order that best meets the desired frequency domain specifications is obtained.

6.5 USING A SOFTWARE PROGRAM TO DESIGN PRACTICAL FIR FILTERS

The DSP56002 Demonstration Software program can be used to design practical, linear phase response FIR filters as follows:

- Boot the PC.
- Insert the DSP56002 Demonstration Software program diskette into Drive "A."
- Type `a:chapter6<return>` to load the DSP56002 Demonstration Software chapter 6 program.
- Select from the screen menus to design the desired FIR filter.
- Decrement (increment) the filter order until the desired frequency domain specifications are obtained.

Note: $h(i)$ for the designed filter are stored on the DSP56002 Demonstration Software program diskette in a data file FIRWxx.DAT, where xx is replaced by LP (Low-Pass filter), HP (High-Pass filter), BP (Band-Stop filter), and BS (Band-Stop filter) as appropriate.

6.5.1 Low-Pass Filter Design Example

User defined specifications:

Sampling frequency	= 48000.000 Hertz
Stopband attenuation	= 50.000 dB
Passband edge	= 8000.000 Hertz
Stopband edge	= 10000.000 Hertz
Window type	= Hamming window
The designed passband attenuation in dB is	.020
The designed stopband attenuation in dB is	52.745

Coefficients of the causal low-pass filter are:

$h(1) = .11213E-02$	$= h(41)$
$h(2) = .13362E-02$	$= h(40)$
$h(3) = .95611E-10$	$= h(39)$
$h(4) = -.23447E-02$	$= h(38)$
$h(5) = -.31894E-02$	$= h(37)$
$h(6) = -.33537E-08$	$= h(36)$
$h(7) = .57094E-02$	$= h(35)$
$h(8) = .74605E-02$	$= h(34)$
$h(9) = .38783E-08$	$= h(33)$
$h(10) = -.12211E-01$	$= h(32)$
$h(11) = -.15372E-01$	$= h(31)$
$h(12) = -.52937E-09$	$= h(30)$
$h(13) = .23965E-01$	$= h(29)$
$h(14) = .29919E-01$	$= h(28)$
$h(15) = .75957E-08$	$= h(27)$
$h(16) = -.48045E-01$	$= h(26)$
$h(17) = -.63144E-01$	$= h(25)$
$h(18) = -.88328E-08$	$= h(24)$
$h(19) = .13488E+00$	$= h(23)$

$$h(20) = .27418\text{E}+00 = h(22)$$

$$h(21) = .33333\text{E}+00 = h(21)$$

6.5.2 High-Pass Filter Design Example

User defined specifications:

Sampling frequency	= 48000.000 Hertz
Stopband attenuation	= 50.000 dB
Passband edge	= 10000.000 Hertz
Stopband edge	= 8000.000 Hertz
Window type	= Blackman window
The designed passband attenuation in dB is	.026
The designed stopband attenuation in dB is	50.312

Coefficients of the causal high-pass filter are:

$$h(1) = -.19380\text{E}-05 = h(53)$$

$$h(2) = -.35399\text{E}-04 = h(52)$$

$$h(3) = .60145\text{E}-10 = h(51)$$

$$h(4) = .21958\text{E}-03 = h(50)$$

$$h(5) = .20330\text{E}-03 = h(49)$$

$$h(6) = -.46795\text{E}-03 = h(48)$$

$$h(7) = -.87685\text{E}-03 = h(47)$$

$$h(8) = .38364\text{E}-03 = h(46)$$

$$h(9) = .20988\text{E}-02 = h(45)$$

$$h(10) = .74902\text{E}-03 = h(44)$$

$$h(11) = -.33817\text{E}-02 = h(43)$$

$$h(12) = -.36583\text{E}-02 = h(42)$$

$$h(13) = .33748\text{E}-02 = h(41)$$

$$h(14) = .83951\text{E}-02 = h(40)$$

$$h(15) = -.30789\text{E}-08 = h(39)$$

$$h(16) = -.13498\text{E}-01 = h(38)$$

$$h(17) = -.87617\text{E}-02 = h(37)$$

$$h(18) = .15473\text{E}-01 = h(36)$$

$$h(19) = .23634\text{E}-01 = h(35)$$

$$h(20) = -.88315\text{E}-02 = h(34)$$

$$h(21) = -.43002E-01 = h(33)$$

$$h(22) = -.14249E-01 = h(32)$$

$$h(23) = .62815E-01 = h(31)$$

$$h(24) = .71223E-01 = h(30)$$

$$h(25) = -.77762E-01 = h(29)$$

$$h(26) = -.30570E+00 = h(28)$$

$$h(27) = .58333E+00 = h(27)$$

6.5.3 Band-Pass Filter Design Example

User defined specifications:

Sampling frequency	= 48000.000 Hertz
Stopband attenuation	= 50.000 dB
Lower passband edge	= 6000.000 Hertz
Upper passband edge	= 12000.000 Hertz
Lower stopband edge	= 4000.000 Hertz
Upper stopband edge	= 14000.000 Hertz
Window type	= Blackman window
The designed passband attenuation in dB is	.021
The designed stopband attenuation in dB is	52.223

Coefficients of the causal band-pass filter are:

$$h(1) = .12388E-10 = h(57)$$

$$h(2) = -.50005E-04 = h(56)$$

$$h(3) = -.85938E-04 = h(55)$$

$$h(4) = .52566E-04 = h(54)$$

$$h(5) = .15195E-10 = h(53)$$

$$h(6) = -.15064E-03 = h(52)$$

$$h(7) = .78036E-03 = h(51)$$

$$h(8) = .19343E-02 = h(50)$$

$$h(9) = .18979E-09 = h(49)$$

$$h(10) = -.37179E-02 = h(48)$$

$$h(11) = -.29173E-02 = h(47)$$

$$h(12) = .11243E-02 = h(46)$$

$$h(13) = .17394E-08 = h(45)$$

$h(14) = -.18634E-02 = h(44)$
 $h(15) = .80464E-02 = h(43)$
 $h(16) = .17209E-01 = h(42)$
 $h(17) = .89654E-09 = h(41)$
 $h(18) = -.26428E-01 = h(40)$
 $h(19) = -.19047E-01 = h(39)$
 $h(20) = .68508E-02 = h(38)$
 $h(21) = .50265E-08 = h(37)$
 $h(22) = -.10397E-01 = h(36)$
 $h(23) = .44256E-01 = h(35)$
 $h(24) = .95863E-01 = h(34)$
 $h(25) = .17518E-08 = h(33)$
 $h(26) = -.17317E+00 = h(32)$
 $h(27) = -.15601E+00 = h(31)$
 $h(28) = .92767E-01 = h(30)$
 $h(29) = .25000E+00 = h(29)$

6.5.4 Band-Stop Filter Design Example

User defined specifications:

Sampling frequency	= 48000.000 Hertz
Stopband attenuation	= 50.000 dB
Lower passband edge	= 4000.000 Hertz
Upper passband edge	= 14000.000 Hertz
Lower stopband edge	= 6000.000 Hertz
Upper stopband edge	= 12000.000 Hertz
Window type	= Hamming window
The designed passband attenuation in dB is	.017
The designed stopband attenuation in dB is	54.043

Coefficients of the causal band-stop filter are:

$h(1) = .52248E-03 = h(47)$
 $h(2) = -.17633E-02 = h(46)$
 $h(3) = -.27286E-02 = h(45)$
 $h(4) = -.12283E-09 = h(44)$

$h(5) = -.64955E-03 = h(43)$
 $h(6) = -.35163E-02 = h(42)$
 $h(7) = .34528E-02 = h(41)$
 $h(8) = .10078E-01 = h(40)$
 $h(9) = .21528E-02 = h(39)$
 $h(10) = .33598E-02 = h(38)$
 $h(11) = .16636E-01 = h(37)$
 $h(12) = .99570E-09 = h(36)$
 $h(13) = -.24860E-01 = h(35)$
 $h(14) = -.75339E-02 = h(34)$
 $h(15) = -.73065E-02 = h(33)$
 $h(16) = -.52450E-01 = h(32)$
 $h(17) = -.28048E-01 = h(31)$
 $h(18) = .45613E-01 = h(30)$
 $h(19) = .13834E-01 = h(29)$
 $h(20) = -.88802E-09 = h(28)$
 $h(21) = .17452E+00 = h(27)$
 $h(22) = .21386E+00 = h(26)$
 $h(23) = -.14770E+00 = h(25)$
 $h(24) = .58333E+00 = h(24)$

6.6 IMPLEMENTING A FIR FILTER

Equation 6.1 is used to implement a FIR filter. In general, the following steps must be performed:

1. Save the input sample $x(n)$ to be the first filter state.
2. Multiply the input sample $x(n)$ by b_0 and accumulate the products of the filter states $x(n-i)$ and the coefficients b_i to yield the output sample $y(n)$.
3. Shift the filter states to obtain the next output sample $y(n+1)$.

6.6.1 Implementing a FIR Filter on the DSP56002 Processor

Steps 1–3 of section 6.6 can be efficiently implemented on the DSP56002 processor by using its:

- A. address pointers to mimic FIFO-like shifting of RAM data
- B. modulo addressing capability to provide wrap-around data buffers

- C. Multiply/Accumulate (MAC) instruction to both multiply two operands and add the product to a third operand in a single instruction cycle
- D. data move capability in parallel with the MAC instruction to keep the multiplier running at 100% capacity
- E. Repeat Next Instruction (REP) instruction to provide compact filter code

The DSP56002 processor's Address Generation Unit is composed of eight Address Registers (R_n), eight associated Offset Registers (N_n), and eight associated Modulo Registers (M_n). The DSP56002 processor's capability to perform modulo addressing allows an Address Register (R_n) value to be incremented (or decremented) and yet remain within an address range of size L , where L is defined by a lower and an upper address boundary.

For the FIR filter, L is equal to the number of coefficients (taps). The value $L-1$ is stored in the DSP56002 processor's Modifier Register (M_n). Because of the manner in which the DSP56002 processor's modulo addressing mechanism is implemented, the lower address boundary of L (base address) must have zeros in its K LSBs, where $2^K \geq L$; e.g., the base address value must be a multiple of 2^K . This requirement is applied to the DSP56002 processor's Address Register (R_n). The upper address boundary is the sum of the lower address boundary (base address) plus the modulo size minus one; e.g., the base address value plus $L-1$. Note, the upper address boundary is not stored in a user visible register.

When modulo addressing is used, the Address Register (R_n) points to a modulo (circular) data buffer located in X-Memory and or Y-Memory. The address pointer (R_n) is not required to point at the lower address boundary; that is, it can point anywhere within the defined modulo address range L . If the address pointer increments past the upper address boundary (base address plus $L-1$ plus 1), it will wrap around to the base address.

6.6.2 DSP56002 Instructions for Implementing a FIR Filter

The DSP56002 instructions shown in Fig. 6.5 are used to implement the FIR filter algorithm where:

1. Modulo Register $M0$ is programmed to the value $NTAPS-1$ (modulo $NTAPS$). Address Register $R0$ is programmed to point to the state variable modulo buffer located in X-Memory. Modulo Register $M4$ is programmed to the value $NTAPS-1$ (modulo $NTAPS$). Address Register $R4$ is programmed to point to the coefficient buffer located in Y-Memory. Given that the FIR filter algorithm has been executing for some time and is ready to process the input sample $x(n)$ in the Data ALU Input Register $X0$, the address in $R4$ is the base address (lower-boundary) of the coefficient buffer. The address in $R0$ is M , where M is greater than or equal to the lower-boundary X-Memory address and less than or equal to the upper-boundary X-Memory address. The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the DSP56002 processor's A-Accumulator and Data ALU Input Registers $X0$ and $Y0$ are as shown in Fig. 6.6.

Fig. 6.5 DSP56002 Assembler Instructions to Implement the FIR Filter Algorithm

```

;
;  FIR  Filter
;
;  x0 = input sample
;  a = output sample
;  ntaps = number of coefficient taps in the filter
;

      move    #states,r0      ;point to filter states
      move    #ntaps-1,m0     ;mod(ntaps)
      move    #coef,r4        ;point to filter coefficients
      move    #ntaps-1,m4     ;mod(ntaps)

      clr     a                x0,x:(r0)+      y:(r4)+,y0
      rep     #ntaps-1
      mac     x0,y0,a          x:(r0)+,x0      y:(r4)+,y0
      macr    x0,y0,a          (r0)-

```

2. The CLR instruction clears the A-Accumulator and simultaneously:

- a. moves the input sample $x(n)$ from the Data ALU's Input Register X0 to the X-Memory location pointed to by Address Register R0
- b. moves the first coefficient from the Y-Memory location pointed to by Address Register R4 to the Data ALU's Input Register Y0

Both Address Registers R0 and R4 are automatically incremented at the end of the CLR instruction (postincremented).

The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A-Accumulator and Data ALU Input Registers X0 and Y0 after execution of the CLR instruction are as shown in Fig. 6.7.

3. The REP instruction regulates execution of NTAPS-1 iterations of the instruction:

```
mac    x0,y0,a      x:(r0)+,x0      y:(r4)+,y0
```

4. The MAC instruction multiplies the filter state variable in X0 by the coefficient in Y0, adds the product to the A-Accumulator, and simultaneously:

- a. moves the next state variable from the X-Memory location pointed to by the Address Register R0 to the Input Register X0
- b. moves the next coefficient from the Y-Memory location pointed to by Address Register R4 to Input Register Y0

Both Address Registers R0 and R4 are automatically incremented at the end of the MAC instruction (postincremented).

The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A-Accumulator and Data ALU Input Registers X0 and Y0 after execution of the first, second, and last MAC instructions are as shown in Figs. 6.8, 6.9, and 6.10, respectively.

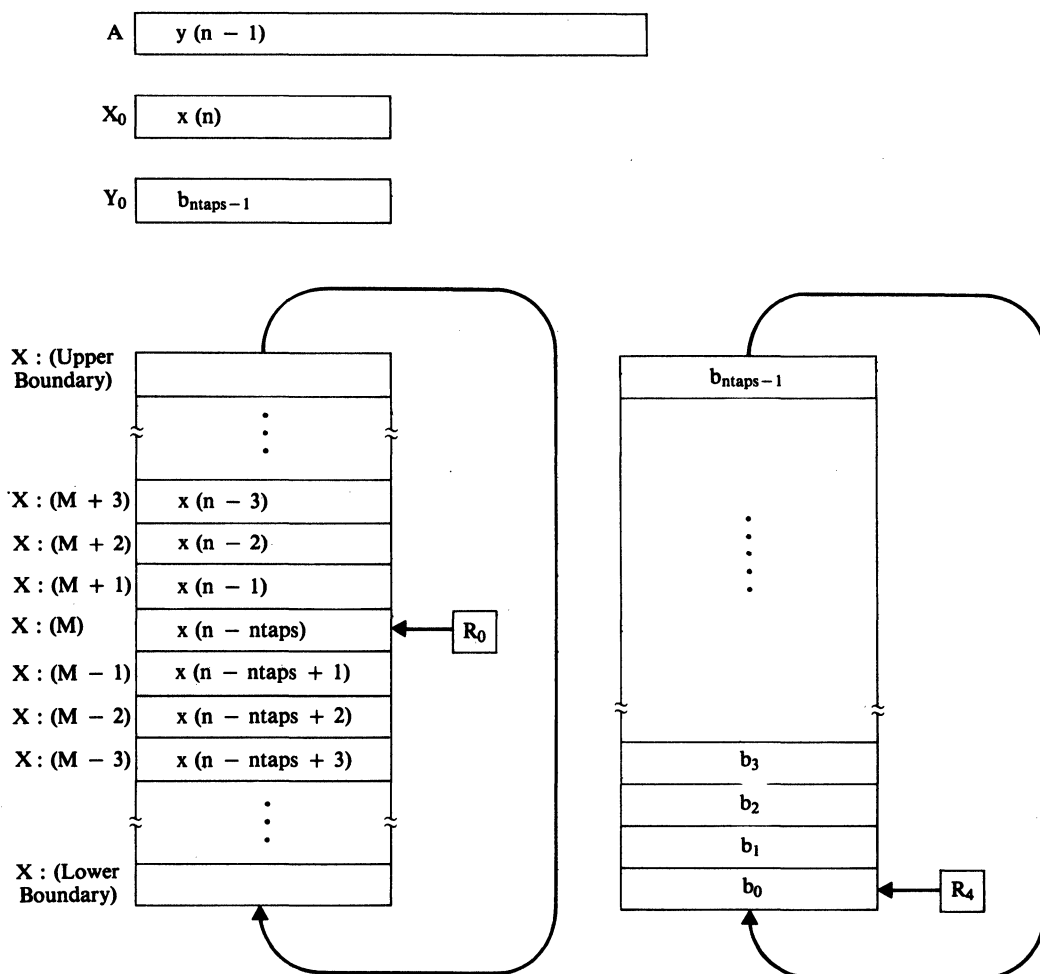


Fig. 6.6 Memory Map and Data Register at the Beginning of the n Iteration

5. The MACR instruction calculates the final tap of the filter algorithm, performs convergent rounding of the result, and simultaneously decrements the Address Register R0 (postdecrementing). At the end of the MACR instruction, the A-Accumulator contains the filter output sample $y(n)$.

Note that during the execution of the filter algorithm, Address Register R4 is postincremented a total of NTAPS times; once in conjunction with the CLR

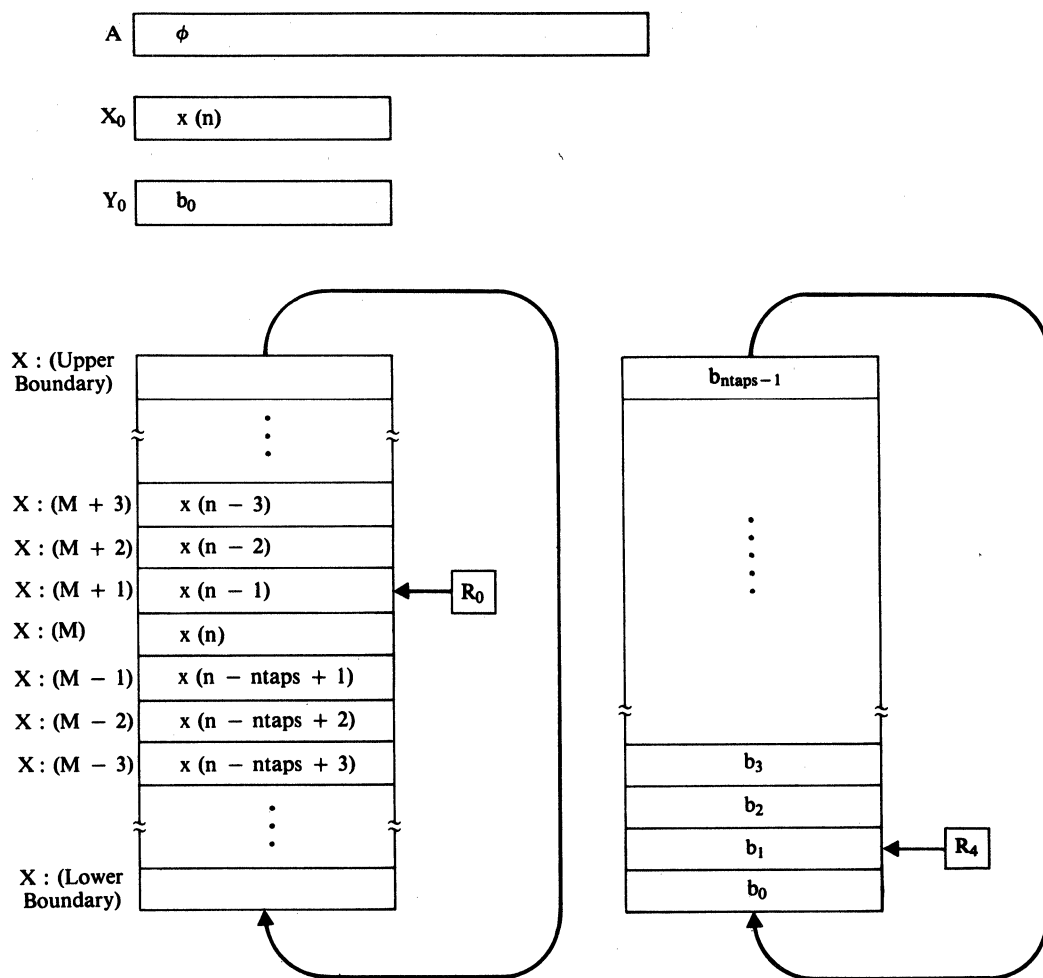


Fig. 6.7 Memory Map and Data Registers after the CLR Iteration

instruction and $NTAPS-1$ times (due to the REP instruction) in conjunction with the MAC instruction. Since the modulus for R4 is $NTAPS$ and R4 is incremented $NTAPS$ times, the address value in R4 wraps around and points to the coefficient buffer's lower boundary location. Recall that the first coefficient b_0 resides in this buffer location.

Note that during the execution of the filter algorithm, Address Register R0 is postincremented a total of $NTAPS$ times; once in conjunction with the CLR

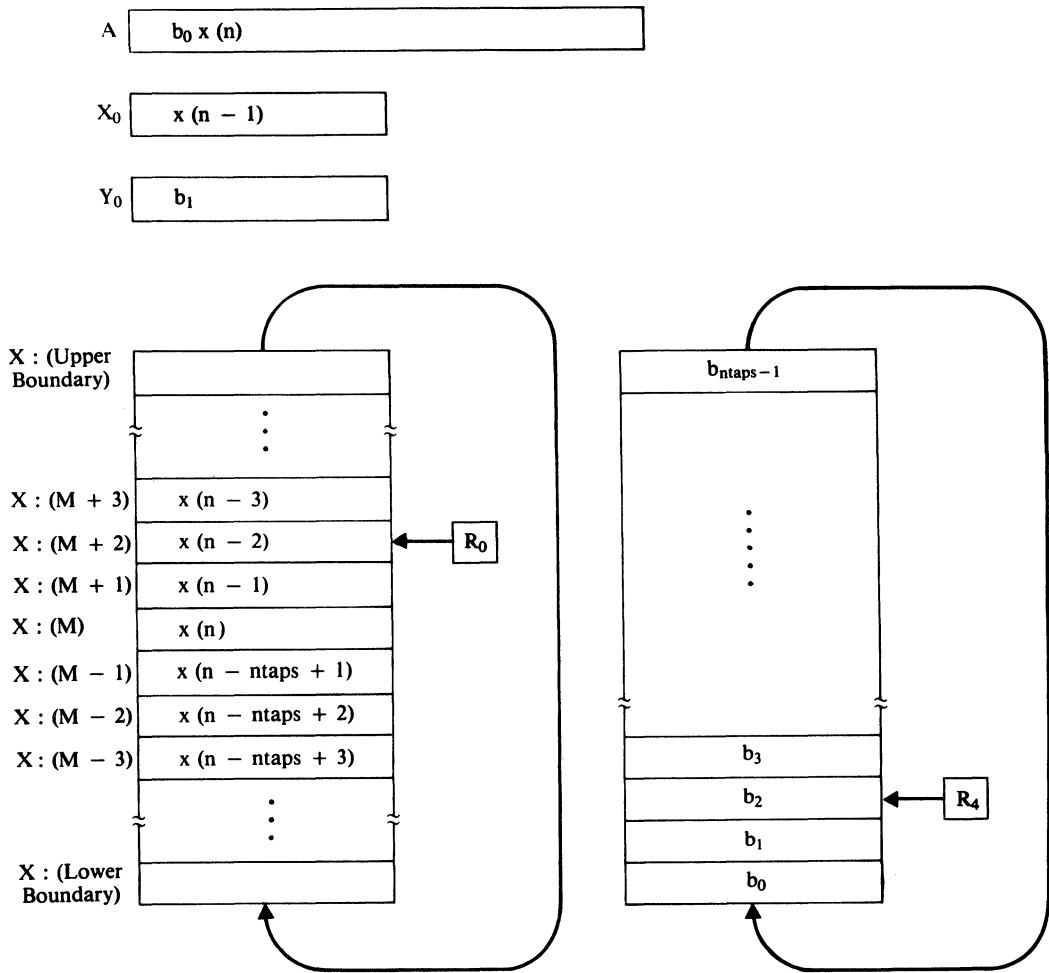


Fig. 6.8 Memory Map and Data Registers after Execution of the First MAC Instruction

instruction and $NTAPS-1$ times (due to the REP instruction) in conjunction with the MAC instruction. Also note that at the beginning of the algorithm, the input sample $x(n)$ is moved from Data ALU Input Register X_0 to the X-Memory location pointed to by R_0 . Since the modulus for R_0 is $NTAPS$ and R_0 is incremented $NTAPS$ times, the address value in R_0 wraps around and points to the state variable buffer's X-Memory location M . In conjunction with the MACR instruction, R_0 is postdecremented by one. As a result, R_0 then points to the state variable

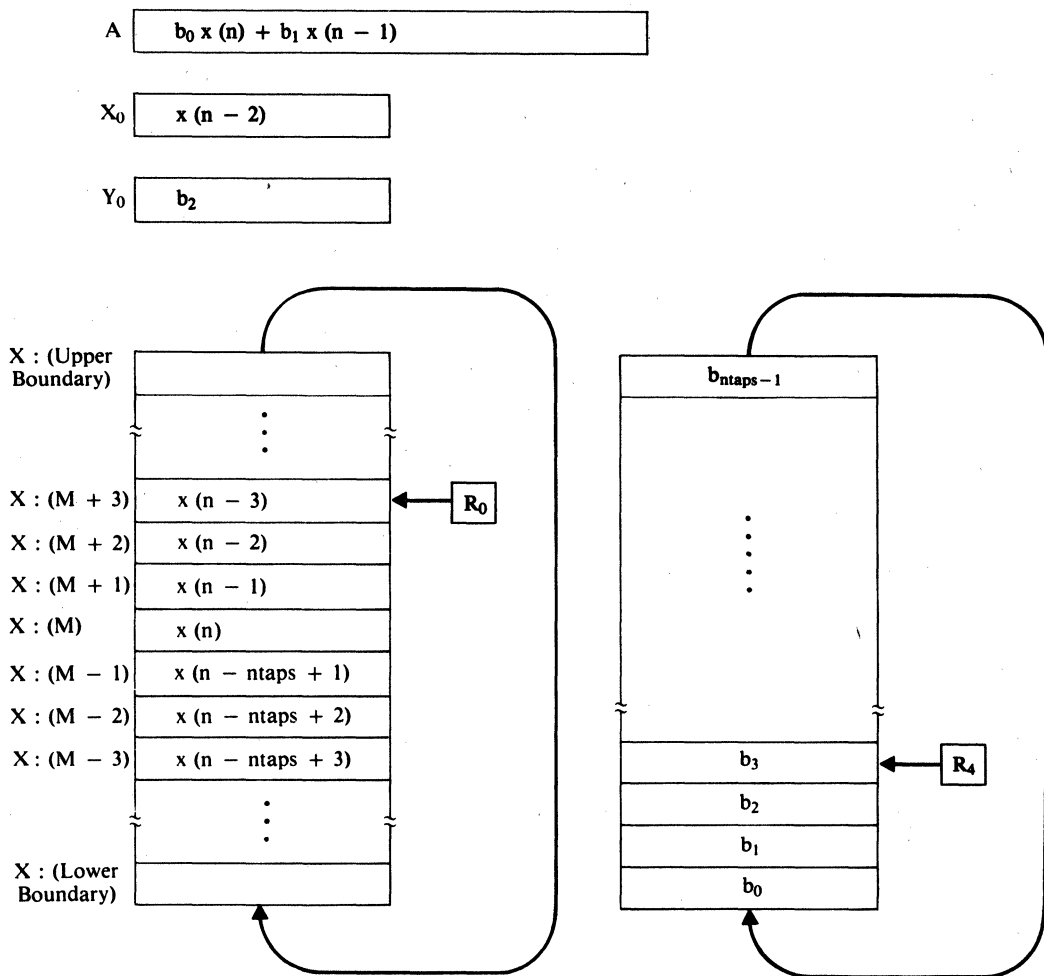


Fig. 6.9 Memory Map and Data Registers after the Execution of the Second MAC Instruction

buffer's X-Memory location M-1. The next time the algorithm is executed, a new (next) input sample $x(n+1)$ will then overwrite the value in X-Memory location M-1. Thus FIFO-like shifting of the filter state variables is accomplished by simply adjusting the R_0 address pointer. The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A-Accumulator and Data ALU Input Registers X_0 and Y_0 after execution of the MACR instruction are as shown in Fig. 6.11.

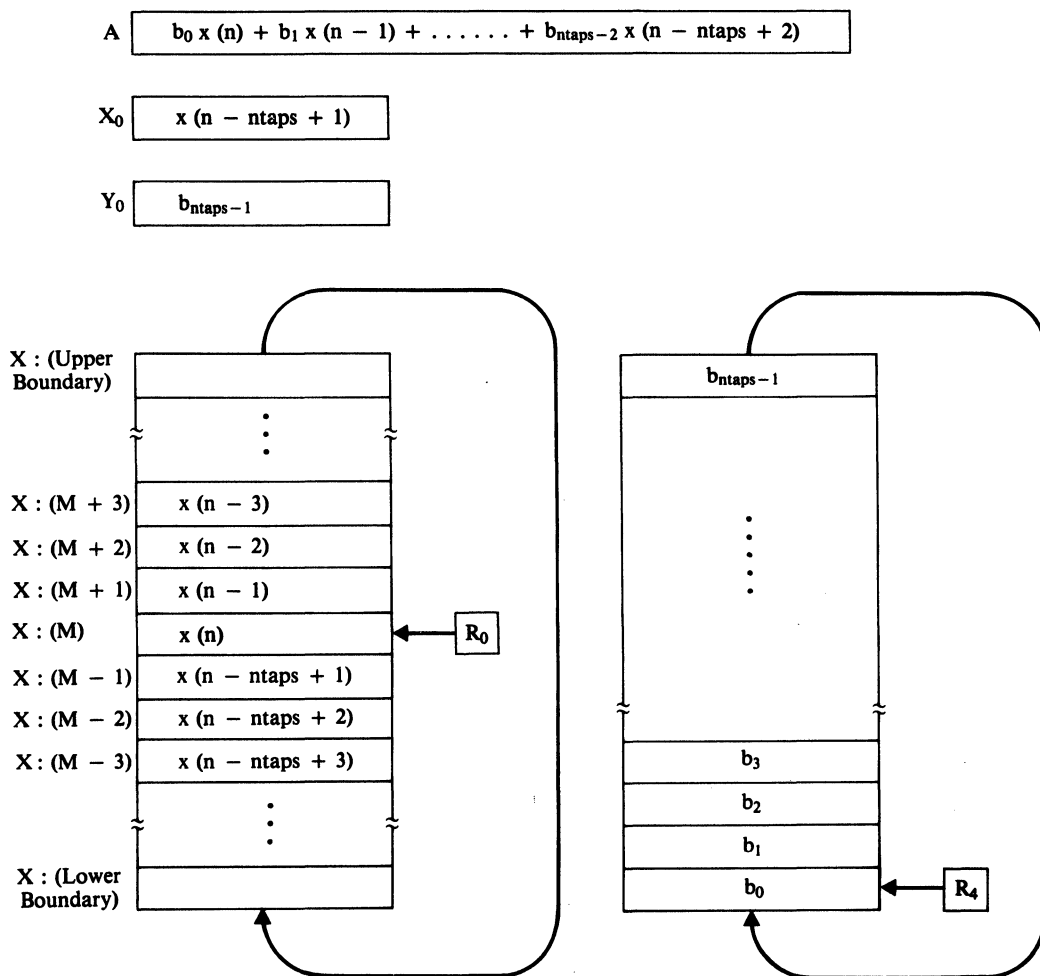


Fig. 6.10 Memory Map and Data Registers after Execution of the Last MAC Instruction

6.6.3 DSP56002 Macros for Implementing a FIR Filter on the DSP System

The two DSP56002 macros shown in Fig. 6.12 process the left channel analog input signal and generate two analog output signals. The left channel analog output signal is the filtered version of the left channel analog input signal. The right channel analog output signal is the left channel analog input signal. The filter of the left channel is implemented using DSP56002 instructions similar to the ones shown in Fig. 6.5. The two DSP56002 macros are included in the STFIR.ASM Assembler Program.

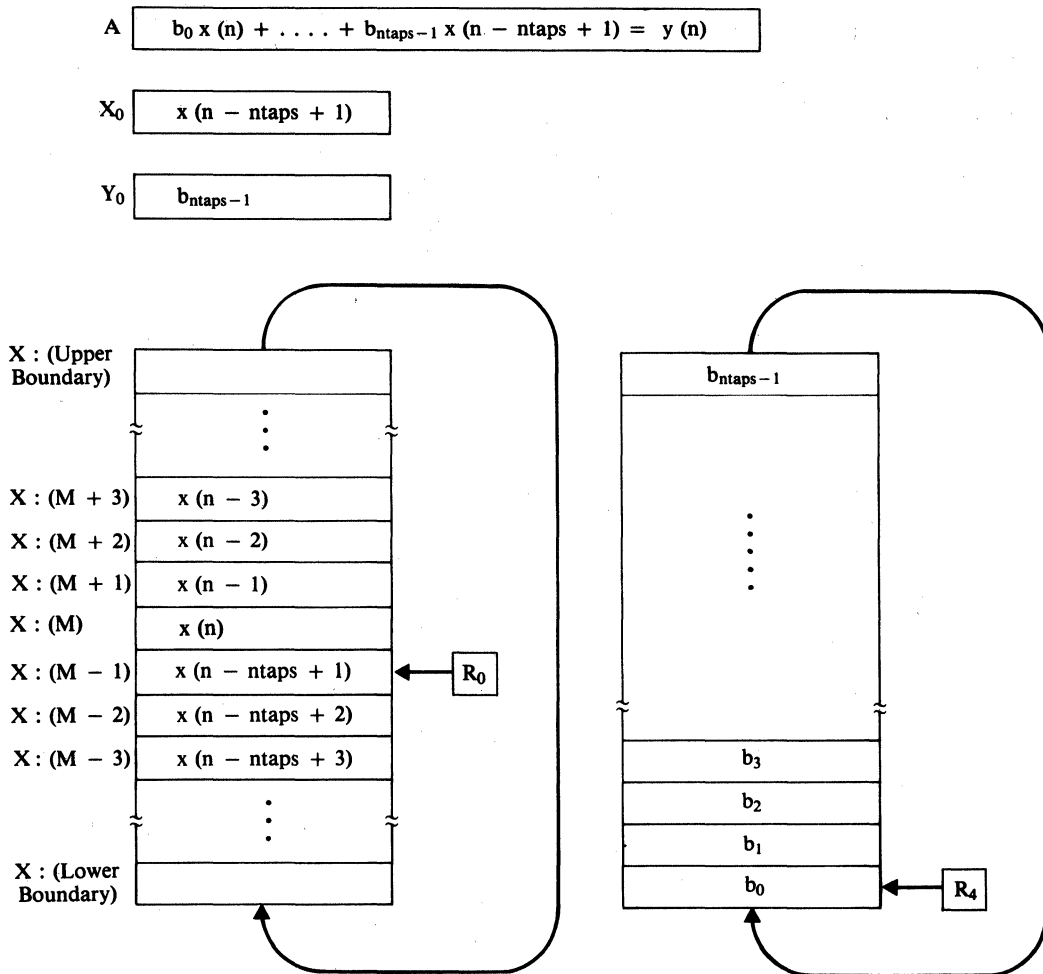


Fig. 6.11 Memory Map and Data Registers after the Execution of the MACR Instruction

Fig. 6.12 STFIR.ASM Assembler Program

```
init_filter macro
; for left channel:
; states_l = Location of filter states in X memory
; coef_l   = Location of filter coefficients in Y memory
; ntaps_l  = Number of taps
;
move    #states_l,r1    ;point to filter states
move    #ntaps_l-1,m1   ;mod(ntaps)
move    #coef_l,r4      ;point to filter coefficients
```

```

        move    #ntaps_1-1,m4    ;mod(ntaps)

    endm

process_filter macro
;*****
; Right Channel (Right Output Signal = Input Signal)
;*****
        move    x0,b
;*****
; Left Channel (Left Output Signal = Output of FIR Filter)
;*****

        clr     a                x0,x:(r1)+    y:(r4)+,y0
        do      #ntaps_1-1,_end
        mac     x0,y0,a          x:(r1)+,x0    y:(r4)+,y0
_end    macr    x0,y0,a          (r1)-
        endm

```

6.7 DIGITAL FILTER DESIGN AND ANALYSIS SYSTEM SOFTWARE PROGRAM

The DSP56002 Demonstration Software Program can be used to design a filter and output the designed filter coefficients. The Digital Filter Design And Analysis System (QEDesign 1000 for Windows) software program by Momentum Data Systems Inc. also is used not only to design filters but also to graphically show the filters' response in both the frequency and time domains. For more information about the QEDesign 1000 software program, please phone Momentum Data Systems Inc., Costa Mesa, California, at (714) 557-6884.

6.8 FIR FILTER DEMONSTRATION SYSTEM

The FIR filtering process can be demonstrated by using the DSP system described in chapter 5 and shown in Fig. 5.1. The DSP system consists of two analog-to-digital (A/D) converters, the DSP56002 processor, and two digital-to-analog (D/A) converters. The DSP56000EVM Evaluation Module (EVM) is used to provide and control the DSP56002 processor, the two A/D converters, and the two D/A converters. The left channel analog input signal $x(t)$ is first digitized using the A/D converter on the EVM. The DSP56002 processor executes the FIR filter algorithm to process the left channel digitized input signal $x(n)$ and generate the left and right channel digital output signals $y_1(n)$ and $y_2(n)$. The left channel digital output signal $y_1(n)$ is the filtered version of the left channel digitized input signal $x(n)$. The right channel digital output signal $y_2(n)$ is the left channel digitized input signal $x(n)$. The two D/A converters on the EVM are then used to convert the left and right channel digital output signals $y_1(n)$ and $y_2(n)$ to the left and right channel analog output signals $y_1(t)$ and $y_2(t)$.

6.9 IMPLEMENTING FIR FILTERS

The **QEDesign 1000 for Windows** program will be used in sections 6.9.1 through 6.9.4 to design low-pass, high-pass, band-pass, and band-stop FIR filters, respectively. The designed filter coefficients, the **STFIR.ASM** assembler program shown in Fig. 6.12 and the **INIT.ASM** assembler program shown in Fig. 5.15, will be used to implement each FIR filter on the DSP system.

6.9.1 Implementing Low-Pass FIR Filter

The following low-pass FIR filter is designed using **QEDesign 1000 for Windows**:

Filter Specifications:

Filter Type:	Low-Pass
Filter Design Method:	FIR Design—Hamming Window
Sampling Frequency:	48000 Hz
Passband Cutoff Frequency:	8000 Hz
Stopband Cutoff Frequency:	10000 Hz
Passband Attenuation:	.5 dB
Stopband Attenuation	50.0 dB

6.9.1.1 Design Results: The following results are obtained using **QEDesign 1000 for Windows**.

FILTER COEFFICIENT FILE FIR DESIGN

```

SAMPLING FREQUENCY      .480000E+05 HERTZ
79                       /* number of taps in decimal      */
004F                     /* number of taps in hexadecimal */
24                       /* number of bits in quantized coefficients (dec) */
0018                     /* number of bits in quantized coefficients (hex) */
0                         /* shift count in decimal      */
0000                     /* shift count in hexadecimal */

      4852  000012F4      /* coefficient of tap      0 */
      3886  00000F2E      /* coefficient of tap      1 */
     -2283  FFFFF715      /* coefficient of tap      2 */
     -6685  FFFFE5E3      /* coefficient of tap      3 */
     -2933  FFFFF48B      /* coefficient of tap      4 */
      6311  000018A7      /* coefficient of tap      5 */
      9688  000025D8      /* coefficient of tap      6 */
           0  00000000      /* coefficient of tap      7 */
     -13472 FFFFCB60      /* coefficient of tap      8 */
     -12135 FFFFD099      /* coefficient of tap      9 */
       7701  00001E15      /* coefficient of tap     10 */
      23497  00005BC9      /* coefficient of tap     11 */
      10449  000028D1      /* coefficient of tap     12 */

```

```

-22333   FFFFA8C3   /* coefficient of tap   13 */
-33595   FFFF7CC5   /* coefficient of tap   14 */
    0     00000000   /* coefficient of tap   15 */
  43967   0000ABBF   /* coefficient of tap   16 */
  38278   00009586   /* coefficient of tap   17 */
-23488   FFFFA440   /* coefficient of tap   18 */
-69394   FFFF0EE     /* coefficient of tap   19 */
-29953   FFFF8AFF   /* coefficient of tap   20 */
  62307   0000F363   /* coefficient of tap   21 */
  91522   00016582   /* coefficient of tap   22 */
    0     00000000   /* coefficient of tap   23 */
-115461  FFFE3CFB   /* coefficient of tap   24 */
-99295   FFFE7C21   /* coefficient of tap   25 */
  60459   0000EC2B   /* coefficient of tap   26 */
 178145   0002B7E1   /* coefficient of tap   27 */
  77131   00012D4B   /* coefficient of tap   28 */
-162027  FFFD8715   /* coefficient of tap   29 */
-242279  FFFC4D99   /* coefficient of tap   30 */
    0     00000000   /* coefficient of tap   31 */
  327231   0004FE3F   /* coefficient of tap   32 */
  298043   00048C3B   /* coefficient of tap   33 */
-196817  FFFCFF2F   /* coefficient of tap   34 */
-651684  FFF60E5C   /* coefficient of tap   35 */
-336041  FFFADF57   /* coefficient of tap   36 */
  938406   000E51A6   /* coefficient of tap   37 */
2463228   002595FC   /* coefficient of tap   38 */
3145728   00300000   /* coefficient of tap   39 */
2463228   002595FC   /* coefficient of tap   40 */
  938406   000E51A6   /* coefficient of tap   41 */
-336041  FFFADF57   /* coefficient of tap   42 */
-651684  FFF60E5C   /* coefficient of tap   43 */
-196817  FFFCFF2F   /* coefficient of tap   44 */
  298043   00048C3B   /* coefficient of tap   45 */
  327231   0004FE3F   /* coefficient of tap   46 */
    0     00000000   /* coefficient of tap   47 */
-242279  FFFC4D99   /* coefficient of tap   48 */
-162027  FFFD8715   /* coefficient of tap   49 */
  77131   00012D4B   /* coefficient of tap   50 */
 178145   0002B7E1   /* coefficient of tap   51 */
  60459   0000EC2B   /* coefficient of tap   52 */
-99295   FFFE7C21   /* coefficient of tap   53 */
-115461  FFFE3CFB   /* coefficient of tap   54 */
    0     00000000   /* coefficient of tap   55 */
  91522   00016582   /* coefficient of tap   56 */
  62307   0000F363   /* coefficient of tap   57 */
-29953   FFFF8AFF   /* coefficient of tap   58 */
-69394   FFFF0EE     /* coefficient of tap   59 */
-23488   FFFFA440   /* coefficient of tap   60 */
  38278   00009586   /* coefficient of tap   61 */
  43967   0000ABBF   /* coefficient of tap   62 */
    0     00000000   /* coefficient of tap   63 */
-33595   FFFF7CC5   /* coefficient of tap   64 */
-22333   FFFFA8C3   /* coefficient of tap   65 */
 10449   000028D1   /* coefficient of tap   66 */

```

```

23497 00005BC9 /* coefficient of tap 67 */
7701 00001E15 /* coefficient of tap 68 */
-12135 FFFFD099 /* coefficient of tap 69 */
-13472 FFFFCB60 /* coefficient of tap 70 */
0 00000000 /* coefficient of tap 71 */
9688 000025D8 /* coefficient of tap 72 */
6311 000018A7 /* coefficient of tap 73 */
-2933 FFFFF48B /* coefficient of tap 74 */
-6685 FFFFE5E3 /* coefficient of tap 75 */
-2283 FFFFF715 /* coefficient of tap 76 */
3886 00000F2E /* coefficient of tap 77 */
4852 000012F4 /* coefficient of tap 78 */

.5784034729003906D-03 3F42F40000000000 /* coefficient of tap 0 */
.4632472991943359D-03 3F3E5C0000000000 /* coefficient of tap 1 */
-.2721548080444336D-03 BF31D60000000000 /* coefficient of tap 2 */
-.7969141006469727D-03 BF4A1D0000000000 /* coefficient of tap 3 */
-.3496408462524414D-03 BF36EA0000000000 /* coefficient of tap 4 */
.7523298263549805D-03 3F48A70000000000 /* coefficient of tap 5 */
.1154899597167969D-02 3F52EC0000000000 /* coefficient of tap 6 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 7 */
-.1605987548828125D-02 BF5A500000000000 /* coefficient of tap 8 */
-.1446604728698730D-02 BF57B38000000000 /* coefficient of tap 9 */
.9180307388305664D-03 3F4E150000000000 /* coefficient of tap 10 */
.2801060676574707D-02 3F66F24000000000 /* coefficient of tap 11 */
.1245617866516113D-02 3F54688000000000 /* coefficient of tap 12 */
-.2662301063537598D-02 BF65CF4000000000 /* coefficient of tap 13 */
-.4004836082458496D-02 BF70676000000000 /* coefficient of tap 14 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 15 */
.5241274833679199D-02 3F7577E000000000 /* coefficient of tap 16 */
.4563093185424805D-02 3F72B0C000000000 /* coefficient of tap 17 */
-.2799987792968750D-02 BF66F00000000000 /* coefficient of tap 18 */
-.8272409439086914D-02 BF80F12000000000 /* coefficient of tap 19 */
-.3570675849914551D-02 BF6D404000000000 /* coefficient of tap 20 */
.7427573204040527D-02 3F7E6C6000000000 /* coefficient of tap 21 */
.1091027259826660D-01 3F86582000000000 /* coefficient of tap 22 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 23 */
-.1376402378082275D-01 BF8C305000000000 /* coefficient of tap 24 */
-.1183688640594482D-01 BF883DF000000000 /* coefficient of tap 25 */
.7207274436950684D-02 3F7D856000000000 /* coefficient of tap 26 */
.2123653888702393D-01 3F95BF0800000000 /* coefficient of tap 27 */
.9194731712341309D-02 3F82D4B000000000 /* coefficient of tap 28 */
-.1931512355804443D-01 BF93C75800000000 /* coefficient of tap 29 */
-.2888190746307373D-01 BF9D933800000000 /* coefficient of tap 30 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 31 */
.3900897502899170D-01 3FA3F8FC00000000 /* coefficient of tap 32 */
.3552949428558350D-01 3FA230EC00000000 /* coefficient of tap 33 */
-.2346241474151611D-01 BF98068800000000 /* coefficient of tap 34 */
-.7768678665161133D-01 BFB3E34800000000 /* coefficient of tap 35 */
-.4005920886993408D-01 BFA482A400000000 /* coefficient of tap 36 */
.1118667125701904D+00 3FBCA34C00000000 /* coefficient of tap 37 */
.2936396598815918D+00 3FD2CAFE00000000 /* coefficient of tap 38 */
.3750000000000000D+00 3FD8000000000000 /* coefficient of tap 39 */

```

```

.2936396598815918D+00    3FD2CAFE00000000    /* coefficient of tap    40 */
.1118667125701904D+00    3FBCA34C00000000    /* coefficient of tap    41 */
-.4005920886993408D-01    BFA482A400000000    /* coefficient of tap    42 */
-.7768678665161133D-01    BFB3E34800000000    /* coefficient of tap    43 */
-.2346241474151611D-01    BF98068800000000    /* coefficient of tap    44 */
.3552949428558350D-01    3FA230EC00000000    /* coefficient of tap    45 */
.3900897502899170D-01    3FA3F8FC00000000    /* coefficient of tap    46 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    47 */
-.2888190746307373D-01    BF9D933800000000    /* coefficient of tap    48 */
-.1931512355804443D-01    BF93C75800000000    /* coefficient of tap    49 */
.9194731712341309D-02    3F82D4B000000000    /* coefficient of tap    50 */
.2123653888702393D-01    3F95BF0800000000    /* coefficient of tap    51 */
.7207274436950684D-02    3F7D856000000000    /* coefficient of tap    52 */
-.1183688640594482D-01    BF883DF000000000    /* coefficient of tap    53 */
-.1376402378082275D-01    BF8C305000000000    /* coefficient of tap    54 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    55 */
.1091027259826660D-01    3F86582000000000    /* coefficient of tap    56 */
.7427573204040527D-02    3F7E6C6000000000    /* coefficient of tap    57 */
-.3570675849914551D-02    BF6D404000000000    /* coefficient of tap    58 */
-.8272409439086914D-02    BF80F12000000000    /* coefficient of tap    59 */
-.2799987792968750D-02    BF66F00000000000    /* coefficient of tap    60 */
.4563093185424805D-02    3F72B0C000000000    /* coefficient of tap    61 */
.5241274833679199D-02    3F7577E000000000    /* coefficient of tap    62 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    63 */
-.4004836082458496D-02    BF70676000000000    /* coefficient of tap    64 */
-.2662301063537598D-02    BF65CF4000000000    /* coefficient of tap    65 */
.1245617866516113D-02    3F54688000000000    /* coefficient of tap    66 */
.2801060676574707D-02    3F66F24000000000    /* coefficient of tap    67 */
.9180307388305664D-03    3F4E150000000000    /* coefficient of tap    68 */
-.1446604728698730D-02    BF57B38000000000    /* coefficient of tap    69 */
-.1605987548828125D-02    BF5A500000000000    /* coefficient of tap    70 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    71 */
.1154899597167969D-02    3F52EC0000000000    /* coefficient of tap    72 */
.7523298263549805D-03    3F48A70000000000    /* coefficient of tap    73 */
-.3496408462524414D-03    BF36EA0000000000    /* coefficient of tap    74 */
-.7969141006469727D-03    BF4A1D0000000000    /* coefficient of tap    75 */
-.2721548080444336D-03    BF31D60000000000    /* coefficient of tap    76 */
.4632472991943359D-03    3F3E5C0000000000    /* coefficient of tap    77 */
.5784034729003906D-03    3F42F40000000000    /* coefficient of tap    78 */

```

Plots of the amplitude response, impulse response, and step response of the designed low-pass FIR filter are shown in Fig. 6.13.

The FIRLP.ASM assembler program, shown in Fig. 6.14, implements the designed low-pass FIR filter on the DSP56002 processor. Note that this program has “include” statements that reference the STFIR.ASM assembler program shown in Fig. 6.12, the INIT.ASM assembler program shown in Fig. 5.15, and the LPCOEF.ASM assembler program shown in Fig. 6.15. A copy of the FIRLP.ASM assembler program, STFIR.ASM assembler program, INIT.ASM assembler program, LPCOEF.ASM assembler program, and the FIRLP.CLD absolute object program can be found on the DSP56002 Demonstration Software program diskette.

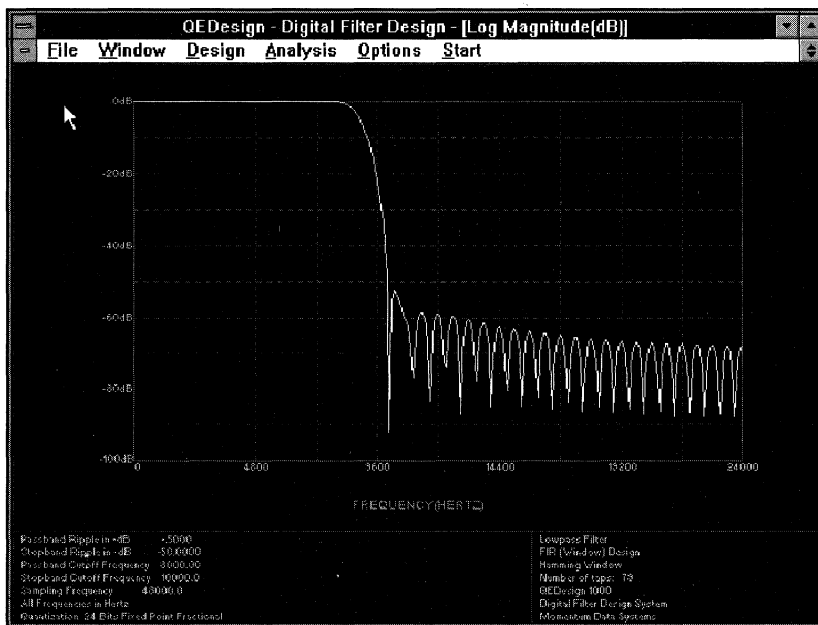


Fig. 6.13 Amplitude Response, Impulse Response, and Step Response of the Designed Low-pass FIR Filter: a) Magnitude

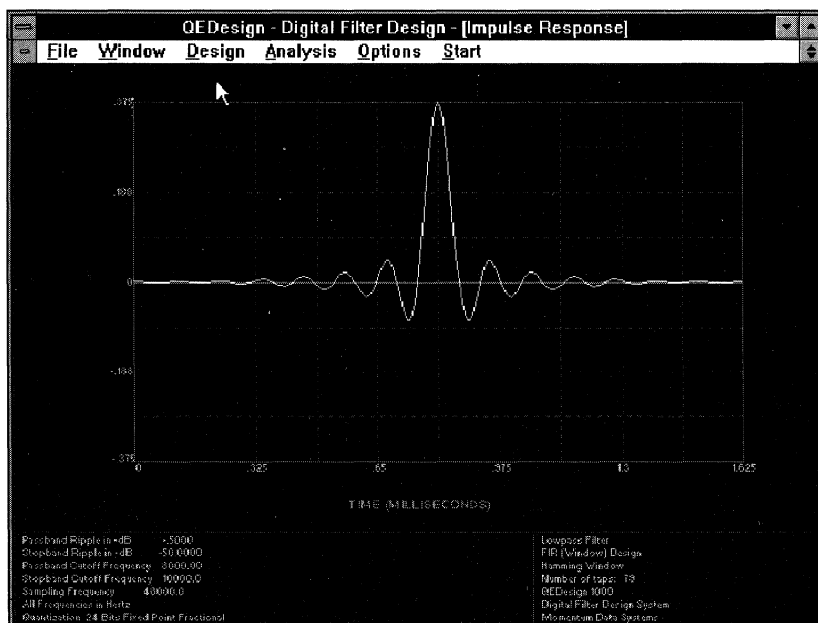


Fig. 6.13 b) Impulse Response

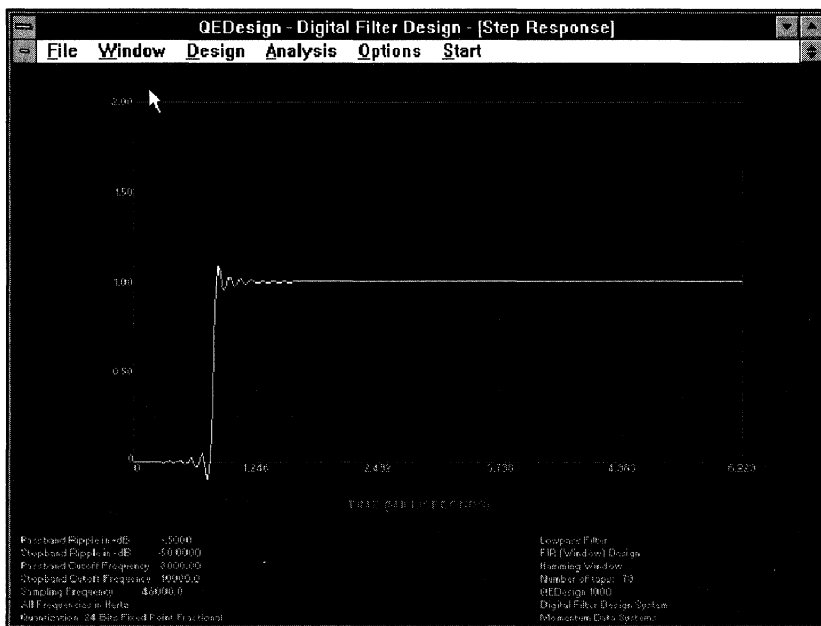


Fig. 6.13 c) Step Response

Fig. 6.14 FIRLP.ASM Assembler Program

```

; right channel output = input signal
; left channel output = output of a low pass filter

include 'init.asm'
include 'lpcoef.asm'
include 'stfir.asm'

init_system
nop

init_filter
nop

data_loop
    wait_receive
    wait_word

    get_left
    get_right

    process_filter

    put_left
    put_right

    jmp data_loop

```


Fig. 6.15 LPCOEF.ASM Assembler Program

```

;*****
;
; File: LPCOEF.ASM
;
;*****
;
;*****
; Coefficients of Low Pass FIR
;*****
;
ntaps_1      equ      79

              org      x:$100
states_1     ds       ntaps_1

              org      y:$100

coef_1
    dc      4852      ; /* coefficient of tap      0 */
    dc      3886      ; /* coefficient of tap      1 */
    dc     -2283      ; /* coefficient of tap      2 */
    dc     -6685      ; /* coefficient of tap      3 */
    dc     -2933      ; /* coefficient of tap      4 */
    dc      6311      ; /* coefficient of tap      5 */
    dc      9688      ; /* coefficient of tap      6 */
    dc         0      ; /* coefficient of tap      7 */
    dc    -13472      ; /* coefficient of tap      8 */
    dc    -12135      ; /* coefficient of tap      9 */
    dc      7701      ; /* coefficient of tap     10 */
    dc      23497      ; /* coefficient of tap     11 */
    dc      10449      ; /* coefficient of tap     12 */
    dc     -22333      ; /* coefficient of tap     13 */
    dc     -33595      ; /* coefficient of tap     14 */
    dc         0      ; /* coefficient of tap     15 */
    dc      43967      ; /* coefficient of tap     16 */
    dc      38278      ; /* coefficient of tap     17 */
    dc     -23488      ; /* coefficient of tap     18 */
    dc     -69394      ; /* coefficient of tap     19 */
    dc     -29953      ; /* coefficient of tap     20 */
    dc      62307      ; /* coefficient of tap     21 */
    dc      91522      ; /* coefficient of tap     22 */
    dc         0      ; /* coefficient of tap     23 */
    dc    -115461      ; /* coefficient of tap     24 */
    dc     -99295      ; /* coefficient of tap     25 */
    dc      60459      ; /* coefficient of tap     26 */
    dc      178145     ; /* coefficient of tap     27 */
    dc      77131      ; /* coefficient of tap     28 */
    dc    -162027      ; /* coefficient of tap     29 */
    dc    -242279      ; /* coefficient of tap     30 */
    dc         0      ; /* coefficient of tap     31 */
    dc      327231     ; /* coefficient of tap     32 */

```

```

dc 298043 ; /* coefficient of tap 33 */
dc -196817 ; /* coefficient of tap 34 */
dc -651684 ; /* coefficient of tap 35 */
dc -336041 ; /* coefficient of tap 36 */
dc 938406 ; /* coefficient of tap 37 */
dc 2463228 ; /* coefficient of tap 38 */
dc 3145728 ; /* coefficient of tap 39 */
dc 2463228 ; /* coefficient of tap 40 */
dc 938406 ; /* coefficient of tap 41 */
dc -336041 ; /* coefficient of tap 42 */
dc -651684 ; /* coefficient of tap 43 */
dc -196817 ; /* coefficient of tap 44 */
dc 298043 ; /* coefficient of tap 45 */
dc 327231 ; /* coefficient of tap 46 */
dc 0 ; /* coefficient of tap 47 */
dc -242279 ; /* coefficient of tap 48 */
dc -162027 ; /* coefficient of tap 49 */
dc 77131 ; /* coefficient of tap 50 */
dc 178145 ; /* coefficient of tap 51 */
dc 60459 ; /* coefficient of tap 52 */
dc -99295 ; /* coefficient of tap 53 */
dc -115461 ; /* coefficient of tap 54 */
dc 0 ; /* coefficient of tap 55 */
dc 91522 ; /* coefficient of tap 56 */
dc 62307 ; /* coefficient of tap 57 */
dc -29953 ; /* coefficient of tap 58 */
dc -69394 ; /* coefficient of tap 59 */
dc -23488 ; /* coefficient of tap 60 */
dc 38278 ; /* coefficient of tap 61 */
dc 43967 ; /* coefficient of tap 62 */
dc 0 ; /* coefficient of tap 63 */
dc -33595 ; /* coefficient of tap 64 */
dc -22333 ; /* coefficient of tap 65 */
dc 10449 ; /* coefficient of tap 66 */
dc 23497 ; /* coefficient of tap 67 */
dc 7701 ; /* coefficient of tap 68 */
dc -12135 ; /* coefficient of tap 69 */
dc -13472 ; /* coefficient of tap 70 */
dc 0 ; /* coefficient of tap 71 */
dc 9688 ; /* coefficient of tap 72 */
dc 6311 ; /* coefficient of tap 73 */
dc -2933 ; /* coefficient of tap 74 */
dc -6685 ; /* coefficient of tap 75 */
dc -2283 ; /* coefficient of tap 76 */
dc 3886 ; /* coefficient of tap 77 */
dc 4852 ; /* coefficient of tap 78 */

```

To demonstrate the designed low-pass FIR filter on the DSP system:

1. Connect a function generator to the left channel of the “in” input on the EVM and to channel one of a dual trace oscilloscope.
2. Set the function generator to produce a 2-volt (peak-to-peak) sine wave.
3. Disconnect the oscilloscope.

0	00000000	/* coefficient of tap	0 */
-7	FFFFFFF9	/* coefficient of tap	1 */
0	00000000	/* coefficient of tap	2 */
72	00000048	/* coefficient of tap	3 */
100	00000064	/* coefficient of tap	4 */
-87	FFFFFFFA9	/* coefficient of tap	5 */
-336	FFFFFFEB0	/* coefficient of tap	6 */
-179	FFFFFFF4D	/* coefficient of tap	7 */
446	000001BE	/* coefficient of tap	8 */
760	000002F8	/* coefficient of tap	9 */
0	00000000	/* coefficient of tap	10 */
-1210	FFFFFFB46	/* coefficient of tap	11 */
-1140	FFFFFFB8C	/* coefficient of tap	12 */
749	000002ED	/* coefficient of tap	13 */
2353	00000931	/* coefficient of tap	14 */
1072	00000430	/* coefficient of tap	15 */
-2338	FFFFFF6DE	/* coefficient of tap	16 */
-3582	FFFFFF202	/* coefficient of tap	17 */
0	00000000	/* coefficient of tap	18 */
4832	000012E0	/* coefficient of tap	19 */
4260	000010A4	/* coefficient of tap	20 */
-2644	FFFFFF5AC	/* coefficient of tap	21 */
-7889	FFFFFFE12F	/* coefficient of tap	22 */
-3433	FFFFFF297	/* coefficient of tap	23 */
7188	00001C14	/* coefficient of tap	24 */
10605	0000296D	/* coefficient of tap	25 */
0	00000000	/* coefficient of tap	26 */
-13399	FFFFCBA9	/* coefficient of tap	27 */
-11478	FFFFD32A	/* coefficient of tap	28 */
6934	00001B16	/* coefficient of tap	29 */
20180	00004ED4	/* coefficient of tap	30 */
8580	00002184	/* coefficient of tap	31 */
-17579	FFFFBB55	/* coefficient of tap	32 */
-25416	FFFF9CB8	/* coefficient of tap	33 */
0	00000000	/* coefficient of tap	34 */
30953	000078E9	/* coefficient of tap	35 */
26080	000065E0	/* coefficient of tap	36 */
-15516	FFFFC364	/* coefficient of tap	37 */
-44513	FFFF521F	/* coefficient of tap	38 */
-18679	FFFFB709	/* coefficient of tap	39 */
37809	000093B1	/* coefficient of tap	40 */
54068	0000D334	/* coefficient of tap	41 */
0	00000000	/* coefficient of tap	42 */
-64645	FFFF037B	/* coefficient of tap	43 */
-54068	FFFF2CCC	/* coefficient of tap	44 */
31973	00007CE5	/* coefficient of tap	45 */
91301	000164A5	/* coefficient of tap	46 */
38196	00009534	/* coefficient of tap	47 */
-77207	FF FED269	/* coefficient of tap	48 */
-110464	FF FE5080	/* coefficient of tap	49 */
0	00000000	/* coefficient of tap	50 */
133092	000207E4	/* coefficient of tap	51 */
112191	0001B63F	/* coefficient of tap	52 */
-67083	FFFFF9F5	/* coefficient of tap	53 */

```

-194446    FFFD0872    /* coefficient of tap    54 */
-82955     FFFEBBF5    /* coefficient of tap    55 */
171972     00029FC4    /* coefficient of tap    56 */
254149     0003E0C5    /* coefficient of tap    57 */
0          00000000    /* coefficient of tap    58 */
-336694     FFFADCCA    /* coefficient of tap    59 */
-304313     FFFB5B47    /* coefficient of tap    60 */
199668     00030BF4    /* coefficient of tap    61 */
657687     000A0917    /* coefficient of tap    62 */
337773     0005276D    /* coefficient of tap    63 */
-940547     FFF1A5FD    /* coefficient of tap    64 */
-2464630     FFDA648A    /* coefficient of tap    65 */
5242879     004FFFFF    /* coefficient of tap    66 */
-2464630     FFDA648A    /* coefficient of tap    67 */
-940547     FFF1A5FD    /* coefficient of tap    68 */
337773     0005276D    /* coefficient of tap    69 */
657687     000A0917    /* coefficient of tap    70 */
199668     00030BF4    /* coefficient of tap    71 */
-304313     FFFB5B47    /* coefficient of tap    72 */
-336694     FFFADCCA    /* coefficient of tap    73 */
0          00000000    /* coefficient of tap    74 */
254149     0003E0C5    /* coefficient of tap    75 */
171972     00029FC4    /* coefficient of tap    76 */
-82955     FFFEBBF5    /* coefficient of tap    77 */
-194446     FFFD0872    /* coefficient of tap    78 */
-67083     FFFEF9F5    /* coefficient of tap    79 */
112191     0001B63F    /* coefficient of tap    80 */
133092     000207E4    /* coefficient of tap    81 */
0          00000000    /* coefficient of tap    82 */
-110464     FFFE5080    /* coefficient of tap    83 */
-77207     FFFED269    /* coefficient of tap    84 */
38196      00009534    /* coefficient of tap    85 */
91301      000164A5    /* coefficient of tap    86 */
31973      00007CE5    /* coefficient of tap    87 */
-54068     FFFF2CCC    /* coefficient of tap    88 */
-64645     FFFF037B    /* coefficient of tap    89 */
0          00000000    /* coefficient of tap    90 */
54068      0000D334    /* coefficient of tap    91 */
37809      000093B1    /* coefficient of tap    92 */
-18679     FFFFB709    /* coefficient of tap    93 */
-44513     FFFF521F    /* coefficient of tap    94 */
-15516     FFFFC364    /* coefficient of tap    95 */
26080      000065E0    /* coefficient of tap    96 */
30953      000078E9    /* coefficient of tap    97 */
0          00000000    /* coefficient of tap    98 */
-25416     FFFF9CB8    /* coefficient of tap    99 */
-17579     FFFFBB55    /* coefficient of tap   100 */
8580       00002184    /* coefficient of tap   101 */
20180      00004ED4    /* coefficient of tap   102 */
6934       00001B16    /* coefficient of tap   103 */
-11478     FFFFD32A    /* coefficient of tap   104 */
-13399     FFFFCBA9    /* coefficient of tap   105 */
0          00000000    /* coefficient of tap   106 */
10605      0000296D    /* coefficient of tap   107 */

```

```

7188      00001C14      /* coefficient of tap 108 */
-3433      FFFFF297      /* coefficient of tap 109 */
-7889      FFFFE12F      /* coefficient of tap 110 */
-2644      FFFFF5AC      /* coefficient of tap 111 */
4260      000010A4      /* coefficient of tap 112 */
4832      000012E0      /* coefficient of tap 113 */
0          00000000      /* coefficient of tap 114 */
-3582      FFFFF202      /* coefficient of tap 115 */
-2338      FFFFF6DE      /* coefficient of tap 116 */
1072      00000430      /* coefficient of tap 117 */
2353      00000931      /* coefficient of tap 118 */
749       000002ED      /* coefficient of tap 119 */
-1140      FFFFFB8C      /* coefficient of tap 120 */
-1210      FFFFFB46      /* coefficient of tap 121 */
0          00000000      /* coefficient of tap 122 */
760       000002F8      /* coefficient of tap 123 */
446       000001BE      /* coefficient of tap 124 */
-179      FFFFFF4D      /* coefficient of tap 125 */
-336      FFFFFEB0      /* coefficient of tap 126 */
-87       FFFFFFFA9      /* coefficient of tap 127 */
100       00000064      /* coefficient of tap 128 */
72        00000048      /* coefficient of tap 129 */
0         00000000      /* coefficient of tap 130 */
-7        FFFFFFFF9      /* coefficient of tap 131 */
0         00000000      /* coefficient of tap 132 */

.0000000000000000D+00      0000000000000000      /* coefficient of tap 0 */
-.8344650268554687D-06      BEAC000000000000      /* coefficient of tap 1 */
.0000000000000000D+00      0000000000000000      /* coefficient of tap 2 */
.8583068847656250D-05      3EE2000000000000      /* coefficient of tap 3 */
.1192092895507812D-04      3EE9000000000000      /* coefficient of tap 4 */
-.1037120819091797D-04      BEE5C00000000000      /* coefficient of tap 5 */
-.4005432128906250D-04      BF05000000000000      /* coefficient of tap 6 */
-.2133846282958984D-04      BEF6600000000000      /* coefficient of tap 7 */
.5316734313964844D-04      3F0BE00000000000      /* coefficient of tap 8 */
.9059906005859375D-04      3F17C00000000000      /* coefficient of tap 9 */
.0000000000000000D+00      0000000000000000      /* coefficient of tap 10 */
-.1442432403564453D-03      BF22E80000000000      /* coefficient of tap 11 */
-.1358985900878906D-03      BF21D00000000000      /* coefficient of tap 12 */
.8928775787353516D-04      3F17680000000000      /* coefficient of tap 13 */
.2804994583129883D-03      3F32620000000000      /* coefficient of tap 14 */
.1277923583984375D-03      3F20C00000000000      /* coefficient of tap 15 */
-.2787113189697266D-03      BF32440000000000      /* coefficient of tap 16 */
-.4270076751708984D-03      BF3BFC0000000000      /* coefficient of tap 17 */
.0000000000000000D+00      0000000000000000      /* coefficient of tap 18 */
.5760192871093750D-03      3F42E00000000000      /* coefficient of tap 19 */
.5078315734863281D-03      3F40A40000000000      /* coefficient of tap 20 */
-.3151893615722656D-03      BF34A80000000000      /* coefficient of tap 21 */
-.9404420852661133D-03      BF4ED10000000000      /* coefficient of tap 22 */
-.4092454910278320D-03      BF3AD20000000000      /* coefficient of tap 23 */
.8568763732910156D-03      3F4C140000000000      /* coefficient of tap 24 */
.1264214515686035D-02      3F54B68000000000      /* coefficient of tap 25 */
.0000000000000000D+00      0000000000000000      /* coefficient of tap 26 */
-.1597285270690918D-02      BF5A2B8000000000      /* coefficient of tap 27 */

```

```

-.1368284225463867D-02    BF566B0000000000    /* coefficient of tap    28 */
.8265972137451172D-03    3F4B160000000000    /* coefficient of tap    29 */
.2405643463134766D-02    3F63B50000000000    /* coefficient of tap    30 */
.1022815704345703D-02    3F50C20000000000    /* coefficient of tap    31 */
-.2095580101013184D-02    BF612AC000000000    /* coefficient of tap    32 */
-.3029823303222656D-02    BF68D20000000000    /* coefficient of tap    33 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    34 */
.3689885139465332D-02    3F6E3A4000000000    /* coefficient of tap    35 */
.3108978271484375D-02    3F69780000000000    /* coefficient of tap    36 */
-.1849651336669922D-02    BF5E4E0000000000    /* coefficient of tap    37 */
-.5306363105773926D-02    BF75BC2000000000    /* coefficient of tap    38 */
-.2226710319519043D-02    BF623DC000000000    /* coefficient of tap    39 */
.4507184028625488D-02    3F72762000000000    /* coefficient of tap    40 */
.6445407867431641D-02    3F7A668000000000    /* coefficient of tap    41 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    42 */
-.7706284523010254D-02    BF7F90A000000000    /* coefficient of tap    43 */
-.6445407867431641D-02    BF7A668000000000    /* coefficient of tap    44 */
.3811478614807129D-02    3F6F394000000000    /* coefficient of tap    45 */
.1088392734527588D-01    3F864A5000000000    /* coefficient of tap    46 */
.4553318023681641D-02    3F72A68000000000    /* coefficient of tap    47 */
-.9203791618347168D-02    BF82D97000000000    /* coefficient of tap    48 */
-.1316833496093750D-01    BF8AF80000000000    /* coefficient of tap    49 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    50 */
.1586580276489258D-01    3F903F2000000000    /* coefficient of tap    51 */
.1337420940399170D-01    3F8B63F000000000    /* coefficient of tap    52 */
-.7996916770935059D-02    BF8060B000000000    /* coefficient of tap    53 */
-.2317976951599121D-01    BF97BC7000000000    /* coefficient of tap    54 */
-.9889006614685059D-02    BF8440B000000000    /* coefficient of tap    55 */
.2050065994262695D-01    3F94FE2000000000    /* coefficient of tap    56 */
.3029692173004150D-01    3F9F062800000000    /* coefficient of tap    57 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    58 */
-.4013705253601074D-01    BFA48CD800000000    /* coefficient of tap    59 */
-.3627693653106689D-01    BFA292E400000000    /* coefficient of tap    60 */
.2380228042602539D-01    3F985FA000000000    /* coefficient of tap    61 */
.7840240001678467D-01    3FB4122E00000000    /* coefficient of tap    62 */
.4026567935943604D-01    3FA49DB400000000    /* coefficient of tap    63 */
-.1121219396591187D+00    BFBCB40600000000    /* coefficient of tap    64 */
-.2938067913055420D+00    BFD2CDBB00000000    /* coefficient of tap    65 */
.6249998807907104D+00    3FE3FFFFC0000000    /* coefficient of tap    66 */
-.2938067913055420D+00    BFD2CDBB00000000    /* coefficient of tap    67 */
-.1121219396591187D+00    BFBCB40600000000    /* coefficient of tap    68 */
.4026567935943604D-01    3FA49DB400000000    /* coefficient of tap    69 */
.7840240001678467D-01    3FB4122E00000000    /* coefficient of tap    70 */
.2380228042602539D-01    3F985FA000000000    /* coefficient of tap    71 */
-.3627693653106689D-01    BFA292E400000000    /* coefficient of tap    72 */
-.4013705253601074D-01    BFA48CD800000000    /* coefficient of tap    73 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    74 */
.3029692173004150D-01    3F9F062800000000    /* coefficient of tap    75 */
.2050065994262695D-01    3F94FE2000000000    /* coefficient of tap    76 */
-.9889006614685059D-02    BF8440B000000000    /* coefficient of tap    77 */
-.2317976951599121D-01    BF97BC7000000000    /* coefficient of tap    78 */
-.7996916770935059D-02    BF8060B000000000    /* coefficient of tap    79 */
.1337420940399170D-01    3F8B63F000000000    /* coefficient of tap    80 */
.1586580276489258D-01    3F903F2000000000    /* coefficient of tap    81 */

```

```

.0000000000000000D+00 0000000000000000 /* coefficient of tap 82 */
-.1316833496093750D-01 BF8AF80000000000 /* coefficient of tap 83 */
-.9203791618347168D-02 BF82D97000000000 /* coefficient of tap 84 */
.4553318023681641D-02 3F72A68000000000 /* coefficient of tap 85 */
.1088392734527588D-01 3F864A5000000000 /* coefficient of tap 86 */
.3811478614807129D-02 3F6F394000000000 /* coefficient of tap 87 */
-.6445407867431641D-02 BF7A668000000000 /* coefficient of tap 88 */
-.7706284523010254D-02 BF7F90A000000000 /* coefficient of tap 89 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 90 */
.6445407867431641D-02 3F7A668000000000 /* coefficient of tap 91 */
.4507184028625488D-02 3F72762000000000 /* coefficient of tap 92 */
-.2226710319519043D-02 BF623DC000000000 /* coefficient of tap 93 */
-.5306363105773926D-02 BF75BC2000000000 /* coefficient of tap 94 */
-.1849651336669922D-02 BF5E4E0000000000 /* coefficient of tap 95 */
.3108978271484375D-02 3F69780000000000 /* coefficient of tap 96 */
.3689885139465332D-02 3F6E3A4000000000 /* coefficient of tap 97 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 98 */
-.3029823303222656D-02 BF68D20000000000 /* coefficient of tap 99 */
-.2095580101013184D-02 BF612AC000000000 /* coefficient of tap 100 */
.1022815704345703D-02 3F50C20000000000 /* coefficient of tap 101 */
.2405643463134766D-02 3F63B50000000000 /* coefficient of tap 102 */
.8265972137451172D-03 3F4B160000000000 /* coefficient of tap 103 */
-.1368284225463867D-02 BF566B0000000000 /* coefficient of tap 104 */
-.1597285270690918D-02 BF5A2B8000000000 /* coefficient of tap 105 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 106 */
.1264214515686035D-02 3F54B68000000000 /* coefficient of tap 107 */
.8568763732910156D-03 3F4C140000000000 /* coefficient of tap 108 */
-.4092454910278320D-03 BF3AD20000000000 /* coefficient of tap 109 */
-.9404420852661133D-03 BF4ED10000000000 /* coefficient of tap 110 */
-.3151893615722656D-03 BF34A80000000000 /* coefficient of tap 111 */
.5078315734863281D-03 3F40A40000000000 /* coefficient of tap 112 */
.5760192871093750D-03 3F42E00000000000 /* coefficient of tap 113 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 114 */
-.4270076751708984D-03 BF3BFC0000000000 /* coefficient of tap 115 */
-.2787113189697266D-03 BF32440000000000 /* coefficient of tap 116 */
.1277923583984375D-03 3F20C00000000000 /* coefficient of tap 117 */
.2804994583129883D-03 3F32620000000000 /* coefficient of tap 118 */
.8928775787353516D-04 3F17680000000000 /* coefficient of tap 119 */
-.1358985900878906D-03 BF21D00000000000 /* coefficient of tap 120 */
-.1442432403564453D-03 BF22E80000000000 /* coefficient of tap 121 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 122 */
.9059906005859375D-04 3F17C00000000000 /* coefficient of tap 123 */
.5316734313964844D-04 3F0BE00000000000 /* coefficient of tap 124 */
-.2133846282958984D-04 BEF6600000000000 /* coefficient of tap 125 */
-.4005432128906250D-04 BF05000000000000 /* coefficient of tap 126 */
-.1037120819091797D-04 BEE5C00000000000 /* coefficient of tap 127 */
.1192092895507812D-04 3EE9000000000000 /* coefficient of tap 128 */
.8583068847656250D-05 3EE2000000000000 /* coefficient of tap 129 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 130 */
-.8344650268554687D-06 BEAC000000000000 /* coefficient of tap 131 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 132 */

```

Plots of the amplitude response, impulse response, and step response of the designed high-pass FIR filter are shown in Fig. 6.16.

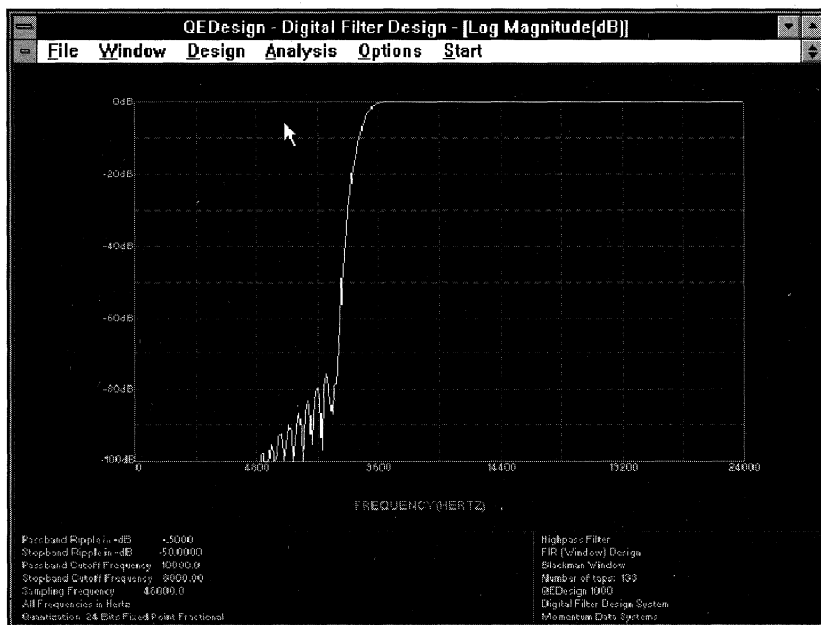


Fig. 6.16 Amplitude Response, Impulse Response, and Step Response of the Designed High-pass FIR Filter: a) Magnitude

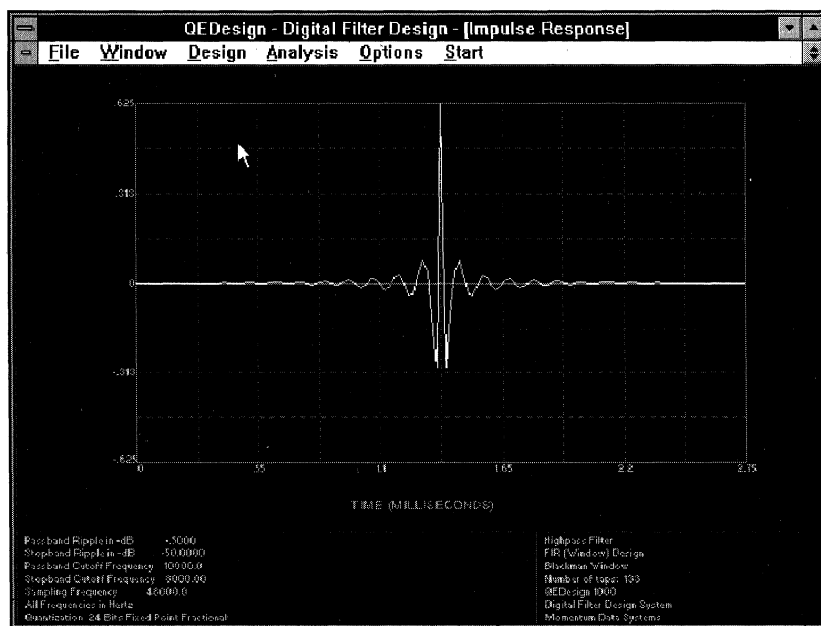


Fig. 6.16 b) Impulse Response

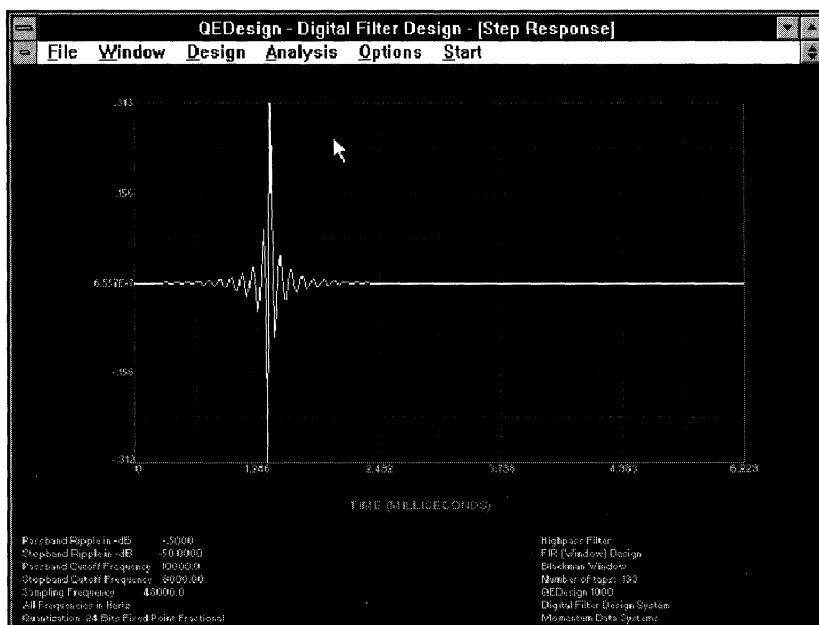


Fig. 6.16 c) Step Response

The FIRHP.ASM assembler program, shown in Fig. 6.17, implements the designed high-pass FIR filter on the DSP56002 processor. Note that this program has “include” statements that reference the STFIR.ASM assembler program shown in Fig. 6.12, the INIT.ASM assembler program shown in Fig. 5.15, and the HPCOE.F.ASM assembler program shown in Fig. 6.18. A copy of the FIRHP.ASM assembler program, STFIR.ASM assembler program, INIT.ASM assembler program, HPCOE.F.ASM assembler program, and the FIRHP.CLD absolute object program can be found on the DSP56002 Demonstration Software program diskette.

Fig. 6.17 FIRHP.ASM Assembler Program

```
; right channel output = input signal
; left channel output = output of a high pass filter

include 'init.asm'
include 'hpcoef.asm'
include 'stfir.asm'

init_system
nop

init_filter
nop

data_loop

wait_receive
```

```

wait_word
get_left
get_right
process_filter
put_left
put_right
jmp data_loop

```

Fig. 6.18 HPCOE.FASM Assembler Program

```

;*****
;
; File: HPCOE.FASM
;
;*****
;
;*****
; Coefficients of High Pass FIR
;*****
;
; number of taps
ntaps_1 equ 79
org x:$100
states_1 ds ntaps_1
org y:$100
coef_1
dc 0 ; /* coefficient of tap 0 */
dc -7 ; /* coefficient of tap 1 */
dc 0 ; /* coefficient of tap 2 */
dc 72 ; /* coefficient of tap 3 */
dc 100 ; /* coefficient of tap 4 */
dc -87 ; /* coefficient of tap 5 */
dc -336 ; /* coefficient of tap 6 */
dc -179 ; /* coefficient of tap 7 */
dc 446 ; /* coefficient of tap 8 */
dc 760 ; /* coefficient of tap 9 */
dc 0 ; /* coefficient of tap 10 */
dc -1210 ; /* coefficient of tap 11 */
dc -1140 ; /* coefficient of tap 12 */
dc 749 ; /* coefficient of tap 13 */
dc 2353 ; /* coefficient of tap 14 */
dc 1072 ; /* coefficient of tap 15 */
dc -2338 ; /* coefficient of tap 16 */
dc -3582 ; /* coefficient of tap 17 */
dc 0 ; /* coefficient of tap 18 */
dc 4832 ; /* coefficient of tap 19 */
dc 4260 ; /* coefficient of tap 20 */

```

```
dc    -2644    ; /* coefficient of tap    21 */
dc    -7889    ; /* coefficient of tap    22 */
dc    -3433    ; /* coefficient of tap    23 */
dc     7188    ; /* coefficient of tap    24 */
dc    10605    ; /* coefficient of tap    25 */
dc     0       ; /* coefficient of tap    26 */
dc   -13399    ; /* coefficient of tap    27 */
dc   -11478    ; /* coefficient of tap    28 */
dc    6934     ; /* coefficient of tap    29 */
dc    20180    ; /* coefficient of tap    30 */
dc    8580     ; /* coefficient of tap    31 */
dc   -17579    ; /* coefficient of tap    32 */
dc   -25416    ; /* coefficient of tap    33 */
dc     0       ; /* coefficient of tap    34 */
dc    30953    ; /* coefficient of tap    35 */
dc    26080    ; /* coefficient of tap    36 */
dc   -15516    ; /* coefficient of tap    37 */
dc   -44513    ; /* coefficient of tap    38 */
dc   -18679    ; /* coefficient of tap    39 */
dc    37809    ; /* coefficient of tap    40 */
dc    54068    ; /* coefficient of tap    41 */
dc     0       ; /* coefficient of tap    42 */
dc   -64645    ; /* coefficient of tap    43 */
dc   -54068    ; /* coefficient of tap    44 */
dc    31973    ; /* coefficient of tap    45 */
dc    91301    ; /* coefficient of tap    46 */
dc    38196    ; /* coefficient of tap    47 */
dc   -77207    ; /* coefficient of tap    48 */
dc  -110464    ; /* coefficient of tap    49 */
dc     0       ; /* coefficient of tap    50 */
dc   133092    ; /* coefficient of tap    51 */
dc   112191    ; /* coefficient of tap    52 */
dc   -67083    ; /* coefficient of tap    53 */
dc  -194446    ; /* coefficient of tap    54 */
dc   -82955    ; /* coefficient of tap    55 */
dc   171972    ; /* coefficient of tap    56 */
dc   254149    ; /* coefficient of tap    57 */
dc     0       ; /* coefficient of tap    58 */
dc  -336694    ; /* coefficient of tap    59 */
dc  -304313    ; /* coefficient of tap    60 */
dc   199668    ; /* coefficient of tap    61 */
dc   657687    ; /* coefficient of tap    62 */
dc   337773    ; /* coefficient of tap    63 */
dc   -940547    ; /* coefficient of tap    64 */
dc  -2464630    ; /* coefficient of tap    65 */
dc   5242879    ; /* coefficient of tap    66 */
dc  -2464630    ; /* coefficient of tap    67 */
dc   -940547    ; /* coefficient of tap    68 */
dc   337773    ; /* coefficient of tap    69 */
dc   657687    ; /* coefficient of tap    70 */
dc   199668    ; /* coefficient of tap    71 */
dc  -304313    ; /* coefficient of tap    72 */
dc  -336694    ; /* coefficient of tap    73 */
dc     0       ; /* coefficient of tap    74 */
```

```
dc 254149 ; /* coefficient of tap 75 */
dc 171972 ; /* coefficient of tap 76 */
dc -82955 ; /* coefficient of tap 77 */
dc -194446 ; /* coefficient of tap 78 */
dc -67083 ; /* coefficient of tap 79 */
dc 112191 ; /* coefficient of tap 80 */
dc 133092 ; /* coefficient of tap 81 */
dc 0 ; /* coefficient of tap 82 */
dc -110464 ; /* coefficient of tap 83 */
dc -77207 ; /* coefficient of tap 84 */
dc 38196 ; /* coefficient of tap 85 */
dc 91301 ; /* coefficient of tap 86 */
dc 31973 ; /* coefficient of tap 87 */
dc -54068 ; /* coefficient of tap 88 */
dc -64645 ; /* coefficient of tap 89 */
dc 0 ; /* coefficient of tap 90 */
dc 54068 ; /* coefficient of tap 91 */
dc 37809 ; /* coefficient of tap 92 */
dc -18679 ; /* coefficient of tap 93 */
dc -44513 ; /* coefficient of tap 94 */
dc -15516 ; /* coefficient of tap 95 */
dc 26080 ; /* coefficient of tap 96 */
dc 30953 ; /* coefficient of tap 97 */
dc 0 ; /* coefficient of tap 98 */
dc -25416 ; /* coefficient of tap 99 */
dc -17579 ; /* coefficient of tap 100 */
dc 8580 ; /* coefficient of tap 101 */
dc 20180 ; /* coefficient of tap 102 */
dc 6934 ; /* coefficient of tap 103 */
dc -11478 ; /* coefficient of tap 104 */
dc -13399 ; /* coefficient of tap 105 */
dc 0 ; /* coefficient of tap 106 */
dc 10605 ; /* coefficient of tap 107 */
dc 7188 ; /* coefficient of tap 108 */
dc -3433 ; /* coefficient of tap 109 */
dc -7889 ; /* coefficient of tap 110 */
dc -2644 ; /* coefficient of tap 111 */
dc 4260 ; /* coefficient of tap 112 */
dc 4832 ; /* coefficient of tap 113 */
dc 0 ; /* coefficient of tap 114 */
dc -3582 ; /* coefficient of tap 115 */
dc -2338 ; /* coefficient of tap 116 */
dc 1072 ; /* coefficient of tap 117 */
dc 2353 ; /* coefficient of tap 118 */
dc 749 ; /* coefficient of tap 119 */
dc -1140 ; /* coefficient of tap 120 */
dc -1210 ; /* coefficient of tap 121 */
dc 0 ; /* coefficient of tap 122 */
dc 760 ; /* coefficient of tap 123 */
dc 446 ; /* coefficient of tap 124 */
dc -179 ; /* coefficient of tap 125 */
dc -336 ; /* coefficient of tap 126 */
dc -87 ; /* coefficient of tap 127 */
dc 100 ; /* coefficient of tap 128 */
```

```

dc      72      ; /* coefficient of tap   129 */
dc      0       ; /* coefficient of tap   130 */
dc     -7       ; /* coefficient of tap   131 */
dc      0       ; /* coefficient of tap   132 */

```

To demonstrate the designed high-pass FIR filter on the DSP system:

1. Connect a function generator to the left channel of the “in” input on the EVM and to channel one of a dual trace oscilloscope.
2. Set the function generator to produce a 2-volt (peak-to-peak) sine wave.
3. Disconnect the oscilloscope.
4. Connect the two channels of the oscilloscope to “out” left and right channel outputs on the EVM.
5. Load the Debug-EVM Software program.
6. Type `change omr 0<return>` to change the contents of the OMR register to 0.
7. From within the Debug-EVM program, load the file `firhp.cld` from the DSP56002 Demonstration Software diskette.
8. Type `go<return>` to execute the designed high-pass FIR filter program (algorithm) on the DSP56002 processor.
9. Vary the sinusoidal input frequency from 0.5 to 20 KHz.

View the left and right channel analog output signals on the oscilloscope. Note that the right channel analog output signal is the left channel analog input signal, and the left channel analog output signal is the filtered version of the left channel analog input signal. Note that the FIR filter stops the frequencies from 0.5 to 8 KHz and passes the frequencies from 10 to 20 KHz as designed. Also note the decreasing attenuation of the left channel analog output signal as the left channel analog input signal’s frequency is increased from 8 to 10 KHz.

10. Type `force b<return>` to halt execution of the FIR filter algorithm and return control of the EVM to its monitor program for command entry.
11. Type `quit<return>` to exit the Debug-EVM program.

6.9.3 Implementing a Band-Pass FIR Filter

The following high-pass FIR filter is designed using QEDesign 1000 for Windows:

Filter Specifications:

Filter Type:	Band-Pass
Filter Design Method:	FIR Design—Blackman Window
Sampling Frequency:	48000 Hz
Passband Cutoff Frequencies:	6000 and 12000 Hz
Stopband Cutoff Frequencies:	4000 and 14000 Hz
Passband Attenuation:	.5 dB
Stopband Attenuation	50.0 dB

6.9.3.1 Design Results: The following results are obtained using QEDesign 1000 for Windows.

FILTER COEFFICIENT FILE FIR DESIGN

```

SAMPLING FREQUENCY      .480000E+05 HERTZ
133                      /* number of taps in decimal      */
0085                     /* number of taps in hexadecimal */
24                       /* number of bits in quantized coefficients (dec) */
0018                    /* number of bits in quantized coefficients (hex) */
0                       /* shift count in decimal      */
0000                    /* shift count in hexadecimal */

    0  00000000          /* coefficient of tap    0 */
    3  00000003          /* coefficient of tap    1 */
   59  0000003B          /* coefficient of tap    2 */
   59  0000003B          /* coefficient of tap    3 */
  -174 FFFFFFF52         /* coefficient of tap    4 */
  -210 FFFFFFF2E         /* coefficient of tap    5 */
    0  00000000          /* coefficient of tap    6 */
  -434 FFFFFFFE4E        /* coefficient of tap    7 */
  -773 FFFFCFB           /* coefficient of tap    8 */
   630 00000276          /* coefficient of tap    9 */
  1815 00000717          /* coefficient of tap   10 */
   501 000001F5          /* coefficient of tap   11 */
    0  00000000          /* coefficient of tap   12 */
  1809 00000711          /* coefficient of tap   13 */
    0  00000000          /* coefficient of tap   14 */
 -5176 FFFFEBC8          /* coefficient of tap   15 */
 -4051 FFFFF02D          /* coefficient of tap   16 */
  1483 000005CB          /* coefficient of tap   17 */
    0  00000000          /* coefficient of tap   18 */
 -2001 FFFFF82F          /* coefficient of tap   19 */
  7379 00001CD3          /* coefficient of tap   20 */
 12767 000031DF          /* coefficient of tap   21 */
    0  00000000          /* coefficient of tap   22 */
 -8288 FFFFDFA0          /* coefficient of tap   23 */
    0  00000000          /* coefficient of tap   24 */
 -4393 FFFFEED7          /* coefficient of tap   25 */
 -22382 FFFFA892         /* coefficient of tap   26 */
 -11100 FFFFD4A4         /* coefficient of tap   27 */
 19881 00004DA9          /* coefficient of tap   28 */
 16742 00004166          /* coefficient of tap   29 */
    0  00000000          /* coefficient of tap   30 */
 20716 000050EC          /* coefficient of tap   31 */
 30449 000076F1          /* coefficient of tap   32 */
 -21055 FFFFADC1         /* coefficient of tap   33 */
 -52630 FFFF326A         /* coefficient of tap   34 */
 -12821 FFFFCDEB         /* coefficient of tap   35 */
    0  00000000          /* coefficient of tap   36 */
 -37459 FFFF6DAD         /* coefficient of tap   37 */
    0  00000000          /* coefficient of tap   38 */
 90191 0001604F          /* coefficient of tap   39 */

```

```

65487  0000FFCF      /* coefficient of tap 40 */
-22395  FFFFA885      /* coefficient of tap 41 */
      0  00000000      /* coefficient of tap 42 */
26776  00006898      /* coefficient of tap 43 */
-93649  FFFE922F      /* coefficient of tap 44 */
-154379 FFFDA4F5      /* coefficient of tap 45 */
      0  00000000      /* coefficient of tap 46 */
92213  00016835      /* coefficient of tap 47 */
      0  00000000      /* coefficient of tap 48 */
45756  0000B2BC      /* coefficient of tap 49 */
227102 0003771E      /* coefficient of tap 50 */
110257 0001AEB1      /* coefficient of tap 51 */
-194322 FFFD08EE      /* coefficient of tap 52 */
-161954 FFFD875E      /* coefficient of tap 53 */
      0  00000000      /* coefficient of tap 54 */
-200272 FFFCF1B0      /* coefficient of tap 55 */
-297864 FFFB7478      /* coefficient of tap 56 */
210543 0003366F      /* coefficient of tap 57 */
544627 00084F73      /* coefficient of tap 58 */
139463 000220C7      /* coefficient of tap 59 */
      0  00000000      /* coefficient of tap 60 */
482042 00075AFA      /* coefficient of tap 61 */
      0  00000000      /* coefficient of tap 62 */
-1630915 FFE71D3D      /* coefficient of tap 63 */
-1629076 FFE7246C      /* coefficient of tap 64 */
1020883 000F93D3      /* coefficient of tap 65 */
2796202 002AAAAA      /* coefficient of tap 66 */
1020883 000F93D3      /* coefficient of tap 67 */
-1629076 FFE7246C      /* coefficient of tap 68 */
-1630915 FFE71D3D      /* coefficient of tap 69 */
      0  00000000      /* coefficient of tap 70 */
482042 00075AFA      /* coefficient of tap 71 */
      0  00000000      /* coefficient of tap 72 */
139463 000220C7      /* coefficient of tap 73 */
544627 00084F73      /* coefficient of tap 74 */
210543 0003366F      /* coefficient of tap 75 */
-297864 FFFB7478      /* coefficient of tap 76 */
-200272 FFFCF1B0      /* coefficient of tap 77 */
      0  00000000      /* coefficient of tap 78 */
-161954 FFFD875E      /* coefficient of tap 79 */
-194322 FFFD08EE      /* coefficient of tap 80 */
110257 0001AEB1      /* coefficient of tap 81 */
227102 0003771E      /* coefficient of tap 82 */
45756  0000B2BC      /* coefficient of tap 83 */
      0  00000000      /* coefficient of tap 84 */
92213  00016835      /* coefficient of tap 85 */
      0  00000000      /* coefficient of tap 86 */
-154379 FFFDA4F5      /* coefficient of tap 87 */
-93649  FFFE922F      /* coefficient of tap 88 */
26776  00006898      /* coefficient of tap 89 */
      0  00000000      /* coefficient of tap 90 */
-22395  FFFFA885      /* coefficient of tap 91 */
65487  0000FFCF      /* coefficient of tap 92 */

```



```

90191 0001604F /* coefficient of tap 93 */
0 00000000 /* coefficient of tap 94 */
-37459 FFFF6DAD /* coefficient of tap 95 */
0 00000000 /* coefficient of tap 96 */
-12821 FFFFCDEB /* coefficient of tap 97 */
-52630 FFFF326A /* coefficient of tap 98 */
-21055 FFFFADG1 /* coefficient of tap 99 */
30449 000076F1 /* coefficient of tap 100 */
20716 000050EC /* coefficient of tap 101 */
0 00000000 /* coefficient of tap 102 */
16742 00004166 /* coefficient of tap 103 */
19881 00004DA9 /* coefficient of tap 104 */
-11100 FFFFD4A4 /* coefficient of tap 105 */
-22382 FFFFA892 /* coefficient of tap 106 */
-4393 FFFFEED7 /* coefficient of tap 107 */
0 00000000 /* coefficient of tap 108 */
-8288 FFFFDFA0 /* coefficient of tap 109 */
0 00000000 /* coefficient of tap 110 */
12767 00Q031DF /* coefficient of tap 111 */
7379 00001CD3 /* coefficient of tap 112 */
-2001 FFFFF82F /* coefficient of tap 113 */
0 00000000 /* coefficient of tap 114 */
1483 000005CB /* coefficient of tap 115 */
-4051 FFFFF02D /* coefficient of tap 116 */
-5176 FFFFEBC8 /* coefficient of tap 117 */
0 00000000 /* coefficient of tap 118 */
1809 00000711 /* coefficient of tap 119 */
0 00000000 /* coefficient of tap 120 */
501 000001F5 /* coefficient of tap 121 */
1815 00000717 /* coefficient of tap 122 */
630 00000276 /* coefficient of tap 123 */
-773 FFFFCFB /* coefficient of tap 124 */
-434 FFFFE4E /* coefficient of tap 125 */
0 00000000 /* coefficient of tap 126 */
-210 FFFFFFF2E /* coefficient of tap 127 */
-174 FFFFFFF52 /* coefficient of tap 128 */
59 0000003B /* coefficient of tap 129 */
59 0000003B /* coefficient of tap 130 */
3 00000003 /* coefficient of tap 131 */
0 00000000 /* coefficient of tap 132 */

.0000000000000000D+00 0000000000000000 /* coefficient of tap 0 */
.3576278686523437D-06 3E98000000000000 /* coefficient of tap 1 */
.7033348083496094D-05 3EDD800000000000 /* coefficient of tap 2 */
.7033348083496094D-05 3EDD800000000000 /* coefficient of tap 3 */
-.2074241638183594D-04 BEF5C00000000000 /* coefficient of tap 4 */
-.2503395080566406D-04 BEFA400000000000 /* coefficient of tap 5 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 6 */
-.5173683166503906D-04 BF0B200000000000 /* coefficient of tap 7 */
-.9214878082275391D-04 BF18280000000000 /* coefficient of tap 8 */
.7510185241699219D-04 3F13B00000000000 /* coefficient of tap 9 */
.2163648605346680D-03 3F2C5C0000000000 /* coefficient of tap 10 */
.5972385406494141D-04 3F0F500000000000 /* coefficient of tap 11 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 12 */

```

```

.2156496047973633D-03    3F2C440000000000    /* coefficient of tap    13 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    14 */
-.6170272827148437D-03    BF44380000000000    /* coefficient of tap    15 */
-.4829168319702148D-03    BF3FA60000000000    /* coefficient of tap    16 */
.1767873764038086D-03    3F272C0000000000    /* coefficient of tap    17 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    18 */
-.2385377883911133D-03    BF2F440000000000    /* coefficient of tap    19 */
.8796453475952148D-03    3F4CD30000000000    /* coefficient of tap    20 */
.1521944999694824D-02    3F58EF8000000000    /* coefficient of tap    21 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    22 */
-.9880065917968750D-03    BF50300000000000    /* coefficient of tap    23 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    24 */
-.5236864089965820D-03    BF41290000000000    /* coefficient of tap    25 */
-.2668142318725586D-02    BF65DB8000000000    /* coefficient of tap    26 */
-.1323223114013672D-02    BF55AE0000000000    /* coefficient of tap    27 */
.2369999885559082D-02    3F636A4000000000    /* coefficient of tap    28 */
.1995801925659180D-02    3F60598000000000    /* coefficient of tap    29 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    30 */
.2469539642333984D-02    3F643B0000000000    /* coefficient of tap    31 */
.3629803657531738D-02    3F6DBC4000000000    /* coefficient of tap    32 */
-.2509951591491699D-02    BF648FC000000000    /* coefficient of tap    33 */
-.6273984909057617D-02    BF79B2C000000000    /* coefficient of tap    34 */
-.1528382301330566D-02    BF590A8000000000    /* coefficient of tap    35 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    36 */
-.4465460777282715D-02    BF724A6000000000    /* coefficient of tap    37 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    38 */
.1075160503387451D-01    3F8604F000000000    /* coefficient of tap    39 */
.7806658744812012D-02    3F7FF9E000000000    /* coefficient of tap    40 */
-.2669692039489746D-02    BF65DEC000000000    /* coefficient of tap    41 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    42 */
.3191947937011719D-02    3F6A260000000000    /* coefficient of tap    43 */
-.1116383075714111D-01    BF86DD1000000000    /* coefficient of tap    44 */
-.1840341091156006D-01    BF92D85800000000    /* coefficient of tap    45 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    46 */
.1099264621734619D-01    3F86835000000000    /* coefficient of tap    47 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    48 */
.5454540252685547D-02    3F76578000000000    /* coefficient of tap    49 */
.2707266807556152D-01    3F9BB8F000000000    /* coefficient of tap    50 */
.1314365863800049D-01    3F8AEB1000000000    /* coefficient of tap    51 */
-.2316498756408691D-01    BF97B89000000000    /* coefficient of tap    52 */
-.1930642127990723D-01    BF93C51000000000    /* coefficient of tap    53 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    54 */
-.2387428283691406D-01    BF98728000000000    /* coefficient of tap    55 */
-.3550815582275391D-01    BFA22E2000000000    /* coefficient of tap    56 */
.2509868144989014D-01    3F99B37800000000    /* coefficient of tap    57 */
.6492459774017334D-01    3FB09EE600000000    /* coefficient of tap    58 */
.1662528514862061D-01    3F91063800000000    /* coefficient of tap    59 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    60 */
.5746388435363770D-01    3FAD6BE800000000    /* coefficient of tap    61 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap    62 */
-.1944202184677124D+00    BFC8E2C300000000    /* coefficient of tap    63 */
-.1942009925842285D+00    BFC8DB9400000000    /* coefficient of tap    64 */
.1216987371444702D+00    3FBF27A600000000    /* coefficient of tap    65 */

```

```

.3333332538604736D+00 3FD5555500000000 /* coefficient of tap 66 */
.1216987371444702D+00 3FBF27A600000000 /* coefficient of tap 67 */
-.1942009925842285D+00 BFC8DB9400000000 /* coefficient of tap 68 */
-.1944202184677124D+00 BFC8E2C300000000 /* coefficient of tap 69 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 70 */
.5746388435363770D-01 3FAD6BE800000000 /* coefficient of tap 71 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 72 */
.1662528514862061D-01 3F91063800000000 /* coefficient of tap 73 */
.6492459774017334D-01 3FB09EE600000000 /* coefficient of tap 74 */
.2509868144989014D-01 3F99B37800000000 /* coefficient of tap 75 */
-.3550815582275391D-01 BFA22E2000000000 /* coefficient of tap 76 */
-.2387428283691406D-01 BF98728000000000 /* coefficient of tap 77 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 78 */
-.1930642127990723D-01 BF93C51000000000 /* coefficient of tap 79 */
-.2316498756408691D-01 BF97B89000000000 /* coefficient of tap 80 */
.1314365863800049D-01 3F8AEB1000000000 /* coefficient of tap 81 */
.2707266807556152D-01 3F9BB8F000000000 /* coefficient of tap 82 */
.5454540252685547D-02 3F76578000000000 /* coefficient of tap 83 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 84 */
.1099264621734619D-01 3F86835000000000 /* coefficient of tap 85 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 86 */
-.1840341091156006D-01 BF92D85800000000 /* coefficient of tap 87 */
-.1116383075714111D-01 BF86DD1000000000 /* coefficient of tap 88 */
.3191947937011719D-02 3F6A260000000000 /* coefficient of tap 89 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 90 */
-.2669692039489746D-02 BF65DEC000000000 /* coefficient of tap 91 */
.7806658744812012D-02 3F7FF9E000000000 /* coefficient of tap 92 */
.1075160503387451D-01 3F8604F000000000 /* coefficient of tap 93 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 94 */
-.4465460777282715D-02 BF724A6000000000 /* coefficient of tap 95 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 96 */
-.1528382301330566D-02 BF590A8000000000 /* coefficient of tap 97 */
-.6273984909057617D-02 BF79B2C000000000 /* coefficient of tap 98 */
-.2509951591491699D-02 BF648FC000000000 /* coefficient of tap 99 */
.3629803657531738D-02 3F6DBC4000000000 /* coefficient of tap 100 */
.2469539642333984D-02 3F643B0000000000 /* coefficient of tap 101 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 102 */
.1995801925659180D-02 3F60598000000000 /* coefficient of tap 103 */
.236999885559082D-02 3F636A4000000000 /* coefficient of tap 104 */
-.1323223114013672D-02 BF55AE0000000000 /* coefficient of tap 105 */
-.2668142318725586D-02 BF65DB8000000000 /* coefficient of tap 106 */
-.5236864089965820D-03 BF41290000000000 /* coefficient of tap 107 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 108 */
-.9880065917968750D-03 BF50300000000000 /* coefficient of tap 109 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 110 */
.1521944999694824D-02 3F58EF8000000000 /* coefficient of tap 111 */
.8796453475952148D-03 3F4CD30000000000 /* coefficient of tap 112 */
-.2385377883911133D-03 BF2F440000000000 /* coefficient of tap 113 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 114 */
.1767873764038086D-03 3F272C0000000000 /* coefficient of tap 115 */
-.4829168319702148D-03 BF3FA60000000000 /* coefficient of tap 116 */
-.6170272827148437D-03 BF44380000000000 /* coefficient of tap 117 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 118 */
.2156496047973633D-03 3F2C440000000000 /* coefficient of tap 119 */

```

```

.0000000000000000D+00    0000000000000000    /* coefficient of tap 120 */
.5972385406494141D-04    3F0F500000000000    /* coefficient of tap 121 */
.2163648605346680D-03    3F2C5C0000000000    /* coefficient of tap 122 */
.7510185241699219D-04    3F13B00000000000    /* coefficient of tap 123 */
-.9214878082275391D-04    BF18280000000000    /* coefficient of tap 124 */
-.5173683166503906D-04    BF0B200000000000    /* coefficient of tap 125 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap 126 */
-.2503395080566406D-04    BEFA400000000000    /* coefficient of tap 127 */
-.2074241638183594D-04    BEF5C00000000000    /* coefficient of tap 128 */
.7033348083496094D-05    3EDD800000000000    /* coefficient of tap 129 */
.7033348083496094D-05    3EDD800000000000    /* coefficient of tap 130 */
.3576278686523437D-06    3E98000000000000    /* coefficient of tap 131 */
.0000000000000000D+00    0000000000000000    /* coefficient of tap 132 */

```

Plots of the amplitude response, impulse response, and step response of the designed band-pass FIR filter are shown in Fig. 6.19.

The FIRBP.ASM assembler program, shown in Fig. 6.20, implements the designed band-pass FIR filter on the DSP56002 processor. Note that this program has “include” statements that reference the STFIR.ASM assembler program shown in Fig. 6.12, the INIT.ASM assembler program shown in Fig. 5.15, and the BPCOE.F.ASM assembler program shown in Fig. 6.21. A copy of the FIRBP.ASM assembler program, STFIR.ASM assembler program, INIT.ASM assembler program, BPCOE.F.ASM assembler program, and the FIRBP.CLD absolute object program can be found on the DSP56002 Demonstration Software program diskette.

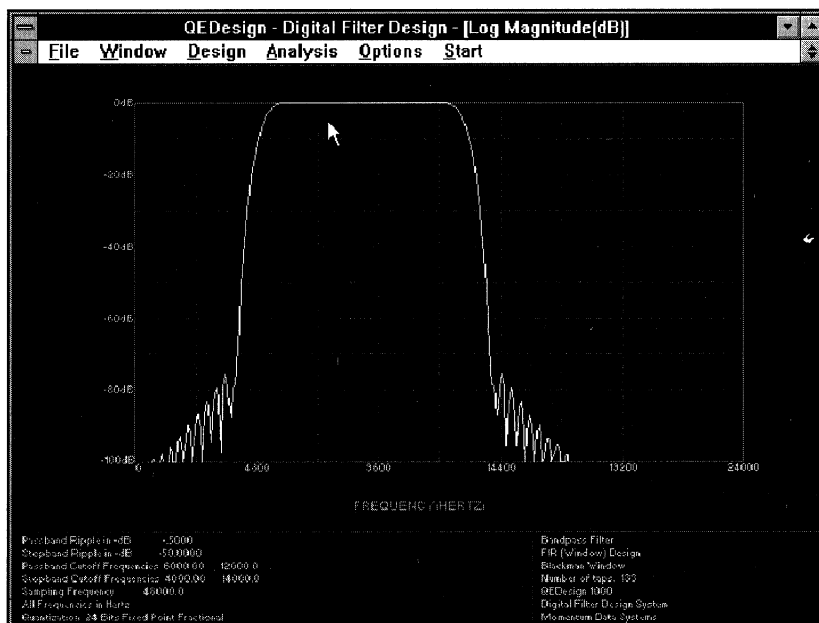


Fig. 6.19 Amplitude Response, Impulse Response, and Step Response of the Designed Band-pass FIR Filter: a) Magnitude

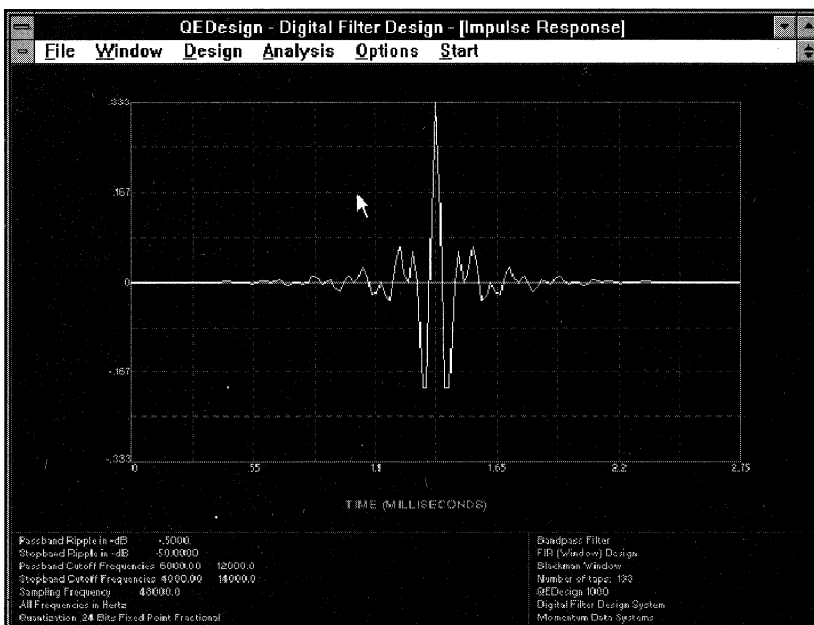


Fig. 6.19 b) Impulse Response



Fig. 6.19 c) Step Response

Fig. 6.20 FIRBPASM Assembler Program

```

; right channel output = input signal
; left channel output = output of a band pass filter

    include 'init.asm'
    include 'bpcoef.asm'
    include 'stfir.asm'

    init_system
    nop

    init_filter
    nop

data_loop
    wait_receive
    wait_word

    get_left
    get_right

    process_filter

    put_left
    put_right

    jmp data_loop

```

Fig. 6.21 BPCOEFSM Assembler Program

```

;*****
;
; File: BPCOEFSM
;
;*****
;
;*****
; Coefficients of bandpass Pass FIR
;*****
;
; number of taps

ntaps_1    equ        133

            org        x:$100
states_1   ds         ntaps_1

            org        y:$100

coef_1
    dc      0          ; /* coefficient of tap    0 */
    dc      3          ; /* coefficient of tap    1 */
    dc     59          ; /* coefficient of tap    2 */

```

```

dc      59      ; /* coefficient of tap      3 */
dc     -174     ; /* coefficient of tap      4 */
dc     -210     ; /* coefficient of tap      5 */
dc       0      ; /* coefficient of tap      6 */
dc     -434     ; /* coefficient of tap      7 */
dc     -773     ; /* coefficient of tap      8 */
dc      630     ; /* coefficient of tap      9 */
dc     1815     ; /* coefficient of tap     10 */
dc      501     ; /* coefficient of tap     11 */
dc       0      ; /* coefficient of tap     12 */
dc     1809     ; /* coefficient of tap     13 */
dc       0      ; /* coefficient of tap     14 */
dc    -5176     ; /* coefficient of tap     15 */
dc   -4051     ; /* coefficient of tap     16 */
dc     1483     ; /* coefficient of tap     17 */
dc       0      ; /* coefficient of tap     18 */
dc    -2001     ; /* coefficient of tap     19 */
dc     7379     ; /* coefficient of tap     20 */
dc    12767     ; /* coefficient of tap     21 */
dc       0      ; /* coefficient of tap     22 */
dc   -8288     ; /* coefficient of tap     23 */
dc       0      ; /* coefficient of tap     24 */
dc   -4393     ; /* coefficient of tap     25 */
dc  -22382     ; /* coefficient of tap     26 */
dc  -11100     ; /* coefficient of tap     27 */
dc   19881     ; /* coefficient of tap     28 */
dc   16742     ; /* coefficient of tap     29 */
dc       0      ; /* coefficient of tap     30 */
dc    20716     ; /* coefficient of tap     31 */
dc   30449     ; /* coefficient of tap     32 */
dc  -21055     ; /* coefficient of tap     33 */
dc  -52630     ; /* coefficient of tap     34 */
dc  -12821     ; /* coefficient of tap     35 */
dc       0      ; /* coefficient of tap     36 */
dc  -37459     ; /* coefficient of tap     37 */
dc       0      ; /* coefficient of tap     38 */
dc   90191     ; /* coefficient of tap     39 */
dc   65487     ; /* coefficient of tap     40 */
dc  -22395     ; /* coefficient of tap     41 */
dc       0      ; /* coefficient of tap     42 */
dc   26776     ; /* coefficient of tap     43 */
dc  -93649     ; /* coefficient of tap     44 */
dc -154379     ; /* coefficient of tap     45 */
dc       0      ; /* coefficient of tap     46 */
dc   92213     ; /* coefficient of tap     47 */
dc       0      ; /* coefficient of tap     48 */
dc   45756     ; /* coefficient of tap     49 */
dc   227102     ; /* coefficient of tap     50 */
dc   110257     ; /* coefficient of tap     51 */
dc  -194322     ; /* coefficient of tap     52 */
dc  -161954     ; /* coefficient of tap     53 */
dc       0      ; /* coefficient of tap     54 */
dc  -200272     ; /* coefficient of tap     55 */
dc  -297864     ; /* coefficient of tap     56 */

```

```
dc 210543 ; /* coefficient of tap 57 */
dc 544627 ; /* coefficient of tap 58 */
dc 139463 ; /* coefficient of tap 59 */
dc 0 ; /* coefficient of tap 60 */
dc 482042 ; /* coefficient of tap 61 */
dc 0 ; /* coefficient of tap 62 */
dc -1630915 ; /* coefficient of tap 63 */
dc -1629076 ; /* coefficient of tap 64 */
dc 1020883 ; /* coefficient of tap 65 */
dc 2796202 ; /* coefficient of tap 66 */
dc 1020883 ; /* coefficient of tap 67 */
dc -1629076 ; /* coefficient of tap 68 */
dc -1630915 ; /* coefficient of tap 69 */
dc 0 ; /* coefficient of tap 70 */
dc 482042 ; /* coefficient of tap 71 */
dc 0 ; /* coefficient of tap 72 */
dc 139463 ; /* coefficient of tap 73 */
dc 544627 ; /* coefficient of tap 74 */
dc 210543 ; /* coefficient of tap 75 */
dc -297864 ; /* coefficient of tap 76 */
dc -200272 ; /* coefficient of tap 77 */
dc 0 ; /* coefficient of tap 78 */
dc -161954 ; /* coefficient of tap 79 */
dc -194322 ; /* coefficient of tap 80 */
dc 110257 ; /* coefficient of tap 81 */
dc 227102 ; /* coefficient of tap 82 */
dc 45756 ; /* coefficient of tap 83 */
dc 0 ; /* coefficient of tap 84 */
dc 92213 ; /* coefficient of tap 85 */
dc 0 ; /* coefficient of tap 86 */
dc -154379 ; /* coefficient of tap 87 */
dc -93649 ; /* coefficient of tap 88 */
dc 26776 ; /* coefficient of tap 89 */
dc 0 ; /* coefficient of tap 90 */
dc -22395 ; /* coefficient of tap 91 */
dc 65487 ; /* coefficient of tap 92 */
dc 90191 ; /* coefficient of tap 93 */
dc 0 ; /* coefficient of tap 94 */
dc -37459 ; /* coefficient of tap 95 */
dc 0 ; /* coefficient of tap 96 */
dc -12821 ; /* coefficient of tap 97 */
dc -52630 ; /* coefficient of tap 98 */
dc -21055 ; /* coefficient of tap 99 */
dc 30449 ; /* coefficient of tap 100 */
dc 20716 ; /* coefficient of tap 101 */
dc 0 ; /* coefficient of tap 102 */
dc 16742 ; /* coefficient of tap 103 */
dc 19881 ; /* coefficient of tap 104 */
dc -11100 ; /* coefficient of tap 105 */
dc -22382 ; /* coefficient of tap 106 */
dc -4393 ; /* coefficient of tap 107 */
dc 0 ; /* coefficient of tap 108 */
dc -8288 ; /* coefficient of tap 109 */
dc 0 ; /* coefficient of tap 110 */
```

```

dc    12767    ; /* coefficient of tap 111 */
dc    7379     ; /* coefficient of tap 112 */
dc   -2001     ; /* coefficient of tap 113 */
dc     0       ; /* coefficient of tap 114 */
dc   1483      ; /* coefficient of tap 115 */
dc  -4051      ; /* coefficient of tap 116 */
dc  -5176      ; /* coefficient of tap 117 */
dc     0       ; /* coefficient of tap 118 */
dc   1809      ; /* coefficient of tap 119 */
dc     0       ; /* coefficient of tap 120 */
dc    501      ; /* coefficient of tap 121 */
dc   1815      ; /* coefficient of tap 122 */
dc    630      ; /* coefficient of tap 123 */
dc   -773      ; /* coefficient of tap 124 */
dc   -434      ; /* coefficient of tap 125 */
dc     0       ; /* coefficient of tap 126 */
dc   -210      ; /* coefficient of tap 127 */
dc   -174      ; /* coefficient of tap 128 */
dc    59       ; /* coefficient of tap 129 */
dc    59       ; /* coefficient of tap 130 */
dc     3       ; /* coefficient of tap 131 */
dc     0       ; /* coefficient of tap 132 */

```

To demonstrate the designed band-pass FIR filter on the DSP system:

1. Connect a function generator to the left channel of the “in” input on the EVM and to channel one of a dual trace oscilloscope.
2. Set the function generator to produce a 2-volt (peak-to-peak) sine wave.
3. Disconnect the oscilloscope.
4. Connect the two channels of the oscilloscope to “out” left and right channel outputs on the EVM.
5. Load the Debug-EVM Software program.
6. Type `change omr 0<return>` to change the contents of the OMR register to 0.
7. From within the Debug-EVM program, load the file **firbp.cld** from the DSP56002 Demonstration Software diskette.
8. Type `go<return>` to execute the designed band-pass FIR filter program (algorithm) on the DSP56002 processor.
9. Vary the sinusoidal input frequency from 0.5 to 20 KHz.

View the left and right channel analog output signals on the oscilloscope. Note that the right channel analog output signal is the left channel analog input signal, and the left channel analog output signal is the filtered version of the left channel analog input signal. Note that the FIR filter passes the frequencies from 6 to 12 KHz and stops the frequencies from 0.5 to 4 KHz and 14 to 20 KHz as designed. Also note the decreasing attenuation of the left channel analog output signal as the left channel analog input signal's frequency is increased from 4 to 6 KHz; note the increasing attenuation of the left channel analog out-

put signal as the left channel analog input signal's frequency is increased from 12 to 14 KHz.

10. Type `force b<return>` to halt execution of the FIR filter algorithm and return control of the EVM to its monitor program for command entry.
11. Type `quit<return>` to exit the Debug-EVM program.

6.9.4 Implementing a Band-Stop FIR Filter

The following band-stop FIR filter is designed using QEDesign 1000 for Windows:

Filter Specifications:

Filter Type:	Band-Stop
Filter Design Method:	FIR Design—Hamming Window
Sampling Frequency:	48000 Hz
Passband Cutoff Frequency:	4000 and 14000 Hz
Stopband Cutoff Frequency:	6000 and 12000 Hz
Passband Attenuation:	.5 dB
Stopband Attenuation	50.0 dB

6.9.4.1 Design Results: The following results are obtained using QEDesign 1000 for Windows.

FILTER COEFFICIENT FILE FIR DESIGN

```

SAMPLING FREQUENCY      .480000E+05 HERTZ
79                        /* number of taps in decimal */
004F                     /* number of taps in hexadecimal */
24                        /* number of bits in quantized coefficients (dec) */
0018                     /* number of bits in quantized coefficients (hex) */
0                         /* shift count in decimal */
0000                     /* shift count in hexadecimal */

      4020 00000FB4      /* coefficient of tap      0 */
     -6731 FFFFE5B5      /* coefficient of tap      1 */
     -5513 FFFFEA77      /* coefficient of tap      2 */
          0 00000000      /* coefficient of tap      3 */
     -7082 FFFFE456      /* coefficient of tap      4 */
    -10932 FFFFD54C      /* coefficient of tap      5 */
       8025 00001F59      /* coefficient of tap      6 */
      21417 000053A9      /* coefficient of tap      7 */
       5580 000015CC      /* coefficient of tap      8 */
          0 00000000      /* coefficient of tap      9 */
      18592 000048A0      /* coefficient of tap     10 */
          0 00000000      /* coefficient of tap     11 */
     -50456 FFFF3AE8      /* coefficient of tap     12 */

```

```

-38682  FFFF68E6  /* coefficient of tap 13 */
13915   0000365B  /* coefficient of tap 14 */
0       00000000  /* coefficient of tap 15 */
-18211  FFFF8BDD  /* coefficient of tap 16 */
66300   000102FC  /* coefficient of tap 17 */
113411  0001BB03  /* coefficient of tap 18 */
0       00000000  /* coefficient of tap 19 */
-72313  FFFEE587  /* coefficient of tap 20 */
0       00000000  /* coefficient of tap 21 */
-37909  FFFF6BEB  /* coefficient of tap 22 */
-192740 FFFD0F1C  /* coefficient of tap 23 */
-95651  FFFE8A5D  /* coefficient of tap 24 */
171984  00029FD0  /* coefficient of tap 25 */
145962  00023A2A  /* coefficient of tap 26 */
0       00000000  /* coefficient of tap 27 */
186212  0002D764  /* coefficient of tap 28 */
280639  0004483F  /* coefficient of tap 29 */
-200711 FFFCEFF9  /* coefficient of tap 30 */
-524585 FFF7FED7  /* coefficient of tap 31 */
-135543 FFFDEE89  /* coefficient of tap 32 */
0       00000000  /* coefficient of tap 33 */
-475158 FFF8BFEA  /* coefficient of tap 34 */
0       00000000  /* coefficient of tap 35 */
1622552 0018C218  /* coefficient of tap 36 */
1625367 0018CD17  /* coefficient of tap 37 */
-1020302 FFF06E72  /* coefficient of tap 38 */
5592405 00555555  /* coefficient of tap 39 */
-1020302 FFF06E72  /* coefficient of tap 40 */
1625367 0018CD17  /* coefficient of tap 41 */
1622552 0018C218  /* coefficient of tap 42 */
0       00000000  /* coefficient of tap 43 */
-475158 FFF8BFEA  /* coefficient of tap 44 */
0       00000000  /* coefficient of tap 45 */
-135543 FFFDEE89  /* coefficient of tap 46 */
-524585 FFF7FED7  /* coefficient of tap 47 */
-200711 FFFCEFF9  /* coefficient of tap 48 */
280639  0004483F  /* coefficient of tap 49 */
186212  0002D764  /* coefficient of tap 50 */
0       00000000  /* coefficient of tap 51 */
145962  00023A2A  /* coefficient of tap 52 */
171984  00029FD0  /* coefficient of tap 53 */
-95651  FFFE8A5D  /* coefficient of tap 54 */
-192740 FFFD0F1C  /* coefficient of tap 55 */
-37909  FFFF6BEB  /* coefficient of tap 56 */
0       00000000  /* coefficient of tap 57 */
-72313  FFFEE587  /* coefficient of tap 58 */
0       00000000  /* coefficient of tap 59 */
113411  0001BB03  /* coefficient of tap 60 */
66300   000102FC  /* coefficient of tap 61 */
-18211  FFFF8BDD  /* coefficient of tap 62 */
0       00000000  /* coefficient of tap 63 */
13915   0000365B  /* coefficient of tap 64 */
-38682  FFFF68E6  /* coefficient of tap 65 */

```

```

-50456 FFFF3AE8 /* coefficient of tap 66 */
      0 00000000 /* coefficient of tap 67 */
18592 000048A0 /* coefficient of tap 68 */
      0 00000000 /* coefficient of tap 69 */
5580 000015CC /* coefficient of tap 70 */
21417 000053A9 /* coefficient of tap 71 */
8025 00001F59 /* coefficient of tap 72 */
-10932 FFFFD54C /* coefficient of tap 73 */
-7082 FFFFE456 /* coefficient of tap 74 */
      0 00000000 /* coefficient of tap 75 */
-5513 FFFFEA77 /* coefficient of tap 76 */
-6731 FFFFE5B5 /* coefficient of tap 77 */
4020 00000FB4 /* coefficient of tap 78 */

.4792213439941406D-03 3F3F680000000000 /* coefficient of tap 0 */
-.8023977279663086D-03 BF4A4B0000000000 /* coefficient of tap 1 */
-.6572008132934570D-03 BF45890000000000 /* coefficient of tap 2 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 3 */
-.8442401885986328D-03 BF4BAA0000000000 /* coefficient of tap 4 */
-.1303195953369141D-02 BF555A0000000000 /* coefficient of tap 5 */
.9566545486450195D-03 3F4F590000000000 /* coefficient of tap 6 */
.2553105354309082D-02 3F64EA4000000000 /* coefficient of tap 7 */
.6651878356933594D-03 3F45CC0000000000 /* coefficient of tap 8 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 9 */
.2216339111328125D-02 3F62280000000000 /* coefficient of tap 10 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 11 */
-.6014823913574219D-02 BF78A30000000000 /* coefficient of tap 12 */
-.4611253738403320D-02 BF72E34000000000 /* coefficient of tap 13 */
.1658797264099121D-02 3F5B2D8000000000 /* coefficient of tap 14 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 15 */
-.2170920372009277D-02 BF61C8C000000000 /* coefficient of tap 16 */
.7903575897216797D-02 3F802FC000000000 /* coefficient of tap 17 */
.1351964473724365D-01 3F8BB03000000000 /* coefficient of tap 18 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 19 */
-.8620381355285645D-02 BF81A79000000000 /* coefficient of tap 20 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 21 */
-.4519104957580566D-02 BF7282A000000000 /* coefficient of tap 22 */
-.2297639846801758D-01 BF97872000000000 /* coefficient of tap 23 */
-.1140248775482178D-01 BF875A3000000000 /* coefficient of tap 24 */
.2050209045410156D-01 3F94FE8000000000 /* coefficient of tap 25 */
.1740002632141113D-01 3F91D15000000000 /* coefficient of tap 26 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 27 */
.2219820022583008D-01 3F96BB2000000000 /* coefficient of tap 28 */
.3345477581024170D-01 3FA120FC00000000 /* coefficient of tap 29 */
-.2392661571502686D-01 BF98803800000000 /* coefficient of tap 30 */
-.6253540515899658D-01 BFB0025200000000 /* coefficient of tap 31 */
-.1615798473358154D-01 BF908BB800000000 /* coefficient of tap 32 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 33 */
-.5664324760437012D-01 BFAD005800000000 /* coefficient of tap 34 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 35 */
.1934232711791992D+00 3FC8C21800000000 /* coefficient of tap 36 */
.1937588453292847D+00 3FC8CD1700000000 /* coefficient of tap 37 */
-.1216704765472412D+00 BFBF231C00000000 /* coefficient of tap 38 */

```

```

.6666666269302368D+00 3FE5555540000000 /* coefficient of tap 39 */
-.1216294765472412D+00 BFBF231C00000000 /* coefficient of tap 40 */
.1937588453292847D+00 3FC8CD1700000000 /* coefficient of tap 41 */
.1934232711791992D+00 3FC8C21800000000 /* coefficient of tap 42 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 43 */
-.5664324760437012D-01 BFAD005800000000 /* coefficient of tap 44 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 45 */
-.1615798473358154D-01 BF908BB800000000 /* coefficient of tap 46 */
-.6253540515899658D-01 BFB0025200000000 /* coefficient of tap 47 */
-.2392661571502686D-01 BF98803800000000 /* coefficient of tap 48 */
.3345477581024170D-01 3FA120FC00000000 /* coefficient of tap 49 */
.2219820022583008D-01 3F96BB2000000000 /* coefficient of tap 50 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 51 */
.1740002632141113D-01 3F91D15000000000 /* coefficient of tap 52 */
.2050209045410156D-01 3F94FE8000000000 /* coefficient of tap 53 */
-.1140248775482178D-01 BF875A3000000000 /* coefficient of tap 54 */
-.2297639846801758D-01 BF97872000000000 /* coefficient of tap 55 */
-.4519104957580566D-02 BF7282A000000000 /* coefficient of tap 56 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 57 */
-.8620381355285645D-02 BF81A79000000000 /* coefficient of tap 58 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 59 */
.1351964473724365D-01 3F8BB03000000000 /* coefficient of tap 60 */
.7903575897216797D-02 3F802FC000000000 /* coefficient of tap 61 */
-.2170920372009277D-02 BF61C8C000000000 /* coefficient of tap 62 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 63 */
.1658797264099121D-02 3F5B2D8000000000 /* coefficient of tap 64 */
-.4611253738403320D-02 BF72E34000000000 /* coefficient of tap 65 */
-.6014823913574219D-02 BF78A30000000000 /* coefficient of tap 66 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 67 */
.2216339111328125D-02 3F62280000000000 /* coefficient of tap 68 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 69 */
.6651878356933594D-03 3F45CC0000000000 /* coefficient of tap 70 */
.2553105354309082D-02 3F64EA4000000000 /* coefficient of tap 71 */
.9566545486450195D-03 3F4F590000000000 /* coefficient of tap 72 */
-.1303195953369141D-02 BF555A0000000000 /* coefficient of tap 73 */
-.8442401885986328D-03 BF4BAA0000000000 /* coefficient of tap 74 */
.0000000000000000D+00 0000000000000000 /* coefficient of tap 75 */
-.6572008132934570D-03 BF45890000000000 /* coefficient of tap 76 */
-.8023977279663086D-03 BF4A4B0000000000 /* coefficient of tap 77 */
.4792213439941406D-03 3F3F680000000000 /* coefficient of tap 78 */

```

Plots of the amplitude response, impulse response, and step response of the designed band-stop FIR filter are shown in Fig. 6.22.

The FIRBS.ASM assembler program, shown in Fig. 6.23, implements the designed band-stop FIR filter on the DSP56002 processor. Note that this program has “include” statements that reference the STFIR.ASM assembler program shown in Fig. 6.12, the INIT.ASM assembler program shown in Fig. 5.15, and the BSCOE.F.ASM assembler program shown in Fig. 6.24. A copy of the FIRBS.ASM assembler program, STFIR.ASM assembler program, INIT.ASM assembler program, BSCOE.F.ASM assembler program, and the FIRBS.CLD absolute object program can be found on the DSP56002 Demonstration Software program diskette.

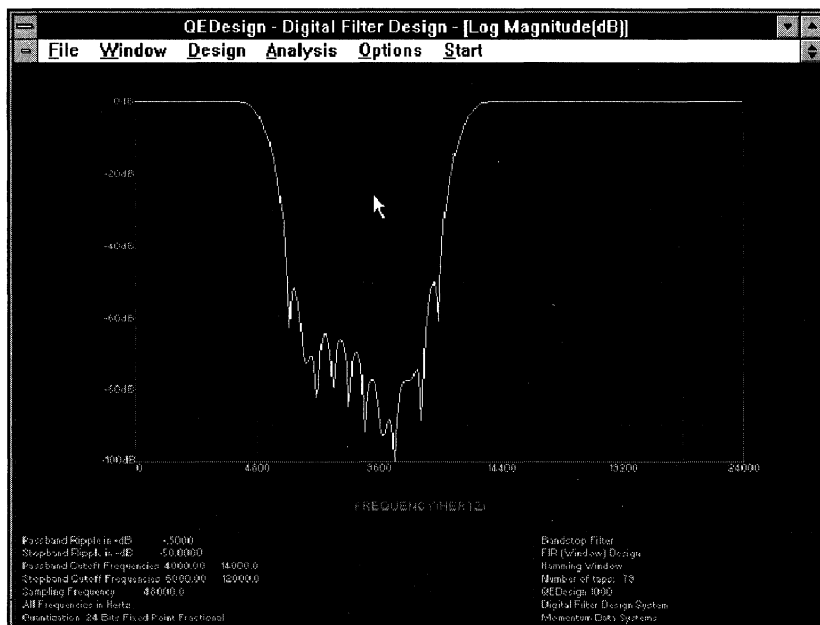


Fig. 6.22 Amplitude Response, Impulse Response, and Step Response of the Designed Band-Stop FIR Filter: a) Magnitude

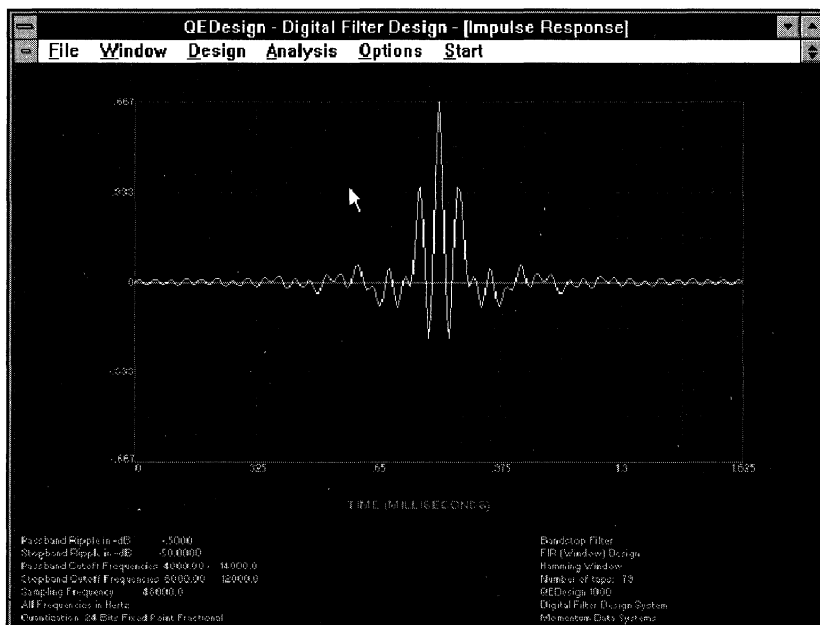


Fig. 6.22 b) Impulse Response

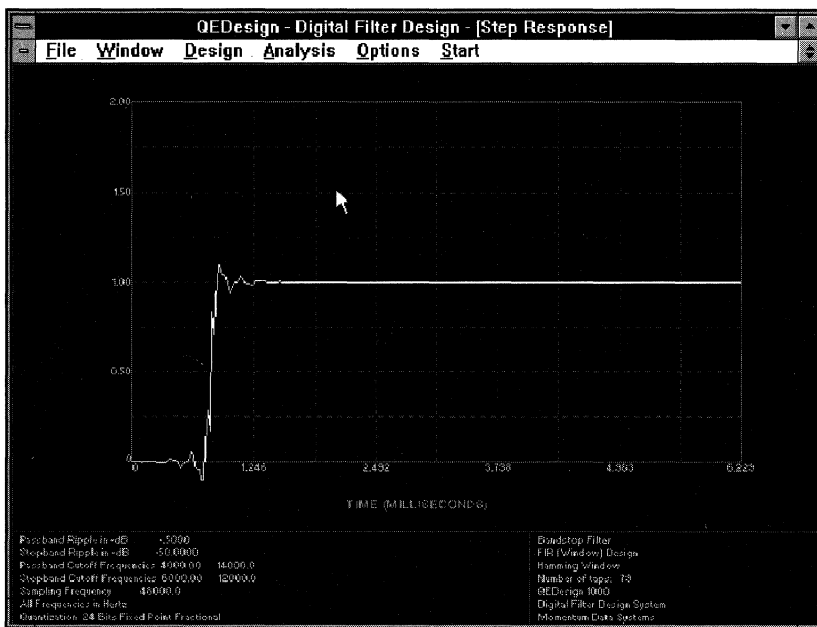


Fig. 6.22 c) Step Response

Fig. 6.23 FIRBS.ASM Assembler Program

```

; right channel output = input signal
; left channel output = output of a band stop filter

include 'init.asm'
include 'bscoef.asm'
include 'stfir.asm'

init_system
nop

init_filter
nop

data_loop

wait_receive
wait_word

get_left
get_right

process_filter

put_left
put_right

jmp data_loop

```

Fig. 6.24 BSCOE.FASM Assembler Program

```

;*****
;
; File: BSCOE.FASM
;
;*****
;
;*****
; Coefficients of bandstop Pass FIR
;*****
;
; number of taps
ntaps_1 equ 79
        org x:$100
states_1 ds ntaps_1
        org y:$100
coef_1
        dc 4020 ; /* coefficient of tap 0 */
        dc -6731 ; /* coefficient of tap 1 */
        dc -5513 ; /* coefficient of tap 2 */
        dc 0 ; /* coefficient of tap 3 */
        dc -7082 ; /* coefficient of tap 4 */
        dc -10932 ; /* coefficient of tap 5 */
        dc 8025 ; /* coefficient of tap 6 */
        dc 21417 ; /* coefficient of tap 7 */
        dc 5580 ; /* coefficient of tap 8 */
        dc 0 ; /* coefficient of tap 9 */
        dc 18592 ; /* coefficient of tap 10 */
        dc 0 ; /* coefficient of tap 11 */
        dc -50456 ; /* coefficient of tap 12 */
        dc -38682 ; /* coefficient of tap 13 */
        dc 13915 ; /* coefficient of tap 14 */
        dc 0 ; /* coefficient of tap 15 */
        dc -18211 ; /* coefficient of tap 16 */
        dc 66300 ; /* coefficient of tap 17 */
        dc 113411 ; /* coefficient of tap 18 */
        dc 0 ; /* coefficient of tap 19 */
        dc -72313 ; /* coefficient of tap 20 */
        dc 0 ; /* coefficient of tap 21 */
        dc -37909 ; /* coefficient of tap 22 */
        dc -192740 ; /* coefficient of tap 23 */
        dc -95651 ; /* coefficient of tap 24 */
        dc 171984 ; /* coefficient of tap 25 */
        dc 145962 ; /* coefficient of tap 26 */
        dc 0 ; /* coefficient of tap 27 */
        dc 186212 ; /* coefficient of tap 28 */
        dc 280639 ; /* coefficient of tap 29 */
        dc -200711 ; /* coefficient of tap 30 */

```



```

dc -524585 ; /* coefficient of tap 31 */
dc -135543 ; /* coefficient of tap 32 */
dc 0 ; /* coefficient of tap 33 */
dc -475158 ; /* coefficient of tap 34 */
dc 0 ; /* coefficient of tap 35 */
dc 1622552 ; /* coefficient of tap 36 */
dc 1625367 ; /* coefficient of tap 37 */
dc -1020302 ; /* coefficient of tap 38 */
dc 5592405 ; /* coefficient of tap 39 */
dc -1020302 ; /* coefficient of tap 40 */
dc 1625367 ; /* coefficient of tap 41 */
dc 1622552 ; /* coefficient of tap 42 */
dc 0 ; /* coefficient of tap 43 */
dc -475158 ; /* coefficient of tap 44 */
dc 0 ; /* coefficient of tap 45 */
dc -135543 ; /* coefficient of tap 46 */
dc -524585 ; /* coefficient of tap 47 */
dc -200711 ; /* coefficient of tap 48 */
dc 280639 ; /* coefficient of tap 49 */
dc 186212 ; /* coefficient of tap 50 */
dc 0 ; /* coefficient of tap 51 */
dc 145962 ; /* coefficient of tap 52 */
dc 171984 ; /* coefficient of tap 53 */
dc -95651 ; /* coefficient of tap 54 */
dc -192740 ; /* coefficient of tap 55 */
dc -37909 ; /* coefficient of tap 56 */
dc 0 ; /* coefficient of tap 57 */
dc -72313 ; /* coefficient of tap 58 */
dc 0 ; /* coefficient of tap 59 */
dc 113411 ; /* coefficient of tap 60 */
dc 66300 ; /* coefficient of tap 61 */
dc -18211 ; /* coefficient of tap 62 */
dc 0 ; /* coefficient of tap 63 */
dc 13915 ; /* coefficient of tap 64 */
dc -38682 ; /* coefficient of tap 65 */
dc -50456 ; /* coefficient of tap 66 */
dc 0 ; /* coefficient of tap 67 */
dc 18592 ; /* coefficient of tap 68 */
dc 0 ; /* coefficient of tap 69 */
dc 5580 ; /* coefficient of tap 70 */
dc 21417 ; /* coefficient of tap 71 */
dc 8025 ; /* coefficient of tap 72 */
dc -10932 ; /* coefficient of tap 73 */
dc -7082 ; /* coefficient of tap 74 */
dc 0 ; /* coefficient of tap 75 */
dc -5513 ; /* coefficient of tap 76 */
dc -6731 ; /* coefficient of tap 77 */
dc 4020 ; /* coefficient of tap 78 */

```

To demonstrate the designed band-stop FIR filter on the DSP system:

1. Connect a function generator to the left channel of the "in" input on the EVM and to channel one of a dual trace oscilloscope.

2. Set the function generator to produce a 2-volt (peak-to-peak) sine wave.
3. Disconnect the oscilloscope.
4. Connect the two channels of the oscilloscope to "out" left and right channel outputs on the EVM.
5. Load the Debug-EVM Software program.
6. Type `change omr 0<return>` to change the contents of the OMR register to 0.
7. From within the Debug-EVM program, load the file **firbs.cld** from the DSP56002 Demonstration Software diskette.
8. Type `go<return>` to execute the designed band-stop FIR filter program (algorithm) on the DSP56002 processor.
9. Vary the sinusoidal input frequency from 0.5 to 20 KHz.

View the left and right channel analog output signals on the oscilloscope. Note that the right channel analog output signal is the left channel analog input signal, and the left channel analog output signal is the filtered version of the left channel analog input signal. Note that the FIR filter stops the frequencies from 6 to 12 KHz and passes the frequencies from 0.5 to 4 KHz and 14 to 20 KHz as designed. Also note the increasing attenuation of the left channel analog output signal as the left channel analog input signal's frequency is increased from 4 to 6 KHz; note the decreasing attenuation of the left channel analog output signal as the left channel analog input signal's frequency is increased from 12 to 14 KHz.

10. Type `force b<return>` to halt execution of the FIR filter algorithm and return control of the EVM to its monitor program for command entry.
11. Type `quit<return>` to exit the Debug-EVM program.

6.10 REFERENCES

1. Alan V. Oppenheim and Ronald W. Shafer, *Discrete-Time Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1989.
2. N.K. Bose, *Digital Filters Theory and Applications*, North-Holland, 1985.
3. Johnny R. Johnson, *Introduction to Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1989.
4. Samuel D. Stearns and Ruth A. David, *Signal Processing Algorithms*, Englewood Cliffs, NJ: Prentice Hall, 1988.
5. Harry Y-F. Lam, *Analog and Digital Filters*, Englewood Cliffs, NJ: Prentice Hall, 1979.
6. Chi-Tsong Chen, *One-Dimensional Digital Signal Processing*, New York: Marcel Dekker, 1979.
7. Richard A. Roberts and Clifford T. Mullis, *Digital Signal Processing*, Reading, MA: Addison-Wesley, 1987.

Designing IIR Filters and Implementing Them on the DSP56002 Processor

In this chapter, Infinite Impulse Response (IIR) digital filters are designed and implemented on the DSP56002 processor. The design of IIR digital filters involves the determination of a transfer function in the z variable that meets certain frequency-domain specifications. The analog-filter-design techniques that have developed over the years are also based on frequency-domain specifications. For this reason, the IIR digital filters are usually designed from analog filters by using various transformations. This approach is most useful for designing standard low-pass, high-pass, band-pass, and band-stop filters, for which a considerable body of knowledge on analog filtering is available. In analog filtering, there are several analog-filter approximations that have been found to be satisfactory in designing analog filters that meet certain frequency-domain specifications. The most popular approximations are the Butterworth and the Chebyshev approximations.

The design of a digital filter begins by designing a suitable low-pass analog filter. Either of two different design procedures are usually used to obtain a digital filter from a low-pass analog filter. When employing the first design procedure, an analog frequency transformation is used to obtain another analog filter of the desired type. The digital filter can be determined from the desired type analog filter by using the Impulse Invariant Transformation or the Bilinear Transformation as explained below. When employing the second design procedure, the low-pass analog filter is converted into a low-pass digital filter by using the Impulse Invariant Transformation or the Bilinear Transformation. A digital frequency transformation is then used to obtain

the digital filter. The stability of the filter then needs to be checked. The first design procedure is considered in this chapter.

The Impulse Invariant Transformation preserves the impulse response of the analog filter by specifying the digital filter's impulse response to be a sampled version of the continuous-time response. As a result, the frequency response of the digital filter is an aliased version of the frequency response of the analog filter. The Bilinear Transformation eliminates the aliasing problem by translating the analog transfer function to a digital transfer function with comparable frequency-response characteristics and by providing a one-to-one mapping of poles and zeros from the continuous time s -plane to the discrete time z -plane. Thus, the Bilinear Transformation is a better choice for designing digital IIR filters that can be implemented on the DSP56002 signal processor.

In the design process, the desired IIR filter's frequency-domain specifications are first used to design normalized low-pass analog Butterworth and Chebyshev filters. Analog frequency transformations are then used to obtain the desired analog filters from the normalized low-pass analog Butterworth and Chebyshev filters. Digital filters are obtained from the analog filters by using the Bilinear Transformation.

By assuming that the filter order is an even integer, the last step in the design process is to decompose the transfer functions of the Butterworth and Chebyshev filters into products of second-order sections. The second-order sections are usually referred to as *biquad* sections. Figure 7.1 shows the procedure that is used in this chapter to design the digital filters. The coefficients of the biquad sections can be used to implement the digital filters on the DSP56002 processor.

In this chapter, the Digital Filter Design and Analysis System (QEDesign 1000 for Windows) software program by Momentum Data Systems, Inc. is used to demonstrate the design process. The DSP56002 instructions that implement a biquad section are described. Low-pass, high-pass, band-pass, and band-stop IIR digital filters are then designed and implemented on the DSP System developed in chapter 5. The DSP56002 instructions that implement the four filter types are included on the DSP56002 Demonstration Software diskette.

7.1 IIR FILTER TRANSFER FUNCTION

The relationship between the IIR filter's input sequence $x(n)$ and the output sequence $y(n)$ can be written as the following difference equation:

$$y(n) = \sum_{i=0}^N b_i x(n-i) + \sum_{j=1}^N a_j y(n-j) \quad (7.1)$$

where b_i and a_j are the filter's coefficients, and at least one of the a_j coefficients is non-zero. The filter coefficients b_i and a_j provide the characteristics of the filter through which the input sequence is passed to create the desired output sequence $y(n)$. By both definition and design, the output sequence $y(n)$ of an IIR filter (also known as a recursive filter) is a function of the past outputs, and the current and past inputs.

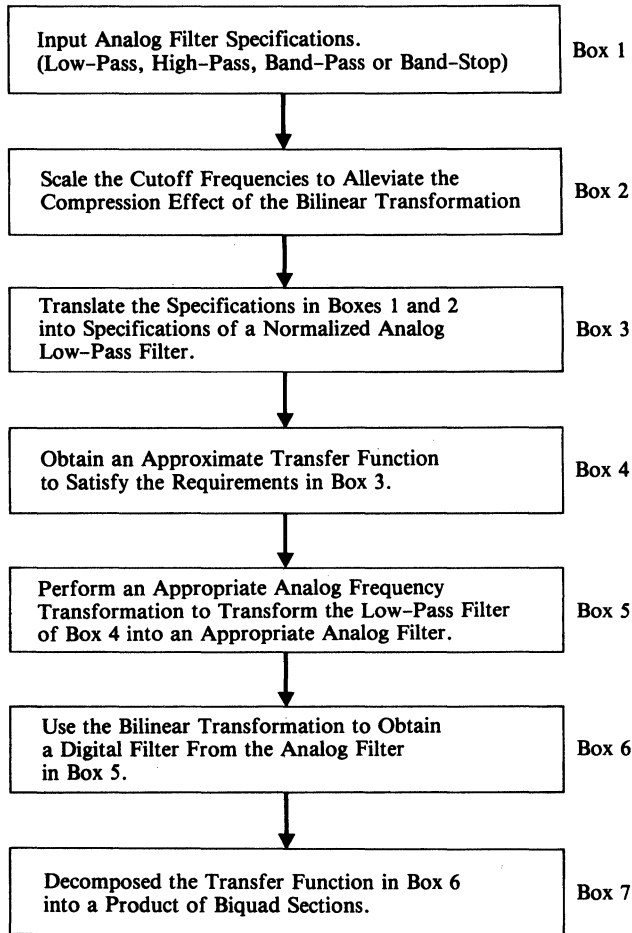


Fig. 7.1 Design Procedure

The transfer function that corresponds to Difference Equation 7.1 can be expressed as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \dots + b_N z^{-N}}{1 - a_1 z^{-1} - \dots - a_N z^{-N}} \quad (7.2)$$

In the remainder of this chapter, N is assumed to be an even integer. The transfer function in Equation 7.2 can then be decomposed into a product of biquad sections:

$$H(z) = K \prod_{i=1}^M H_i(z) \quad (7.3)$$

where

$$H_i(z) = \frac{Y_{i+1}(z)}{Y_i(z)} = \frac{1 + b_{1i}z^{-1} + b_{2i}z^{-2}}{1 - a_{1i}z^{-1} - a_{2i}z^{-2}} \quad (7.4)$$

$$Y_1(z) = KX(z) \quad (7.5)$$

$$Y_{M+1}(z) = Y(z) \quad (7.6)$$

and M is equal to $0.5N$, K is a positive gain, and the output of the i^{th} biquad section $y_{i+1}(n)$ is the input of $i+1$ th biquad section. Figure 7.2 shows the realization of the transfer function $H(z)$ as a cascade of biquad sections. To guarantee the stability of the IIR digital filter, the magnitude of a_{1i} and a_{2i} must be less than 2.0 and 1.0, respectively. The biquad section $H_i(z)$ can be rewritten as:

$$H_i(z) = \frac{Y_{i+1}(z)}{Y_i(z)} = \frac{(1 + b_{1i}z^{-1} + b_{2i}z^{-2})X_i(z)}{(1 - a_{1i}z^{-1} - a_{2i}z^{-2})X_i(z)} \quad (7.7)$$

where $X_i(z)$ is chosen so that:

$$Y_i(z) = (1 - a_{1i}z^{-1} - a_{2i}z^{-2})X_i(z) \quad (7.8)$$

$$Y_{i+1}(z) = (1 + b_{1i}z^{-1} + b_{2i}z^{-2})X_i(z) \quad (7.9)$$

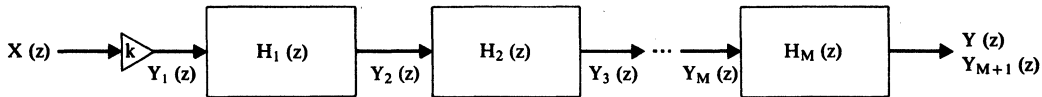


Fig. 7.2 Realization of the Transfer Function as a Cascade of Biquad Sections

The realization of Equations 7.8 and 7.9 is shown in Fig. 7.3. The realization of the transfer function of a sixth order digital IIR filter as a cascade of three biquad sections is shown in Fig. 7.4.

The main objective of this chapter is to design the transfer function $H(z)$ of equations 7.3 and 7.4 to meet certain frequency-domain specifications, and then use the filter coefficients of Equation 7.4 to implement the desired digital filter on the DSP56002 processor.

7.2 REVIEW OF THE BILINEAR TRANSFORMATION

The Bilinear Transformation is defined by the following substitution:

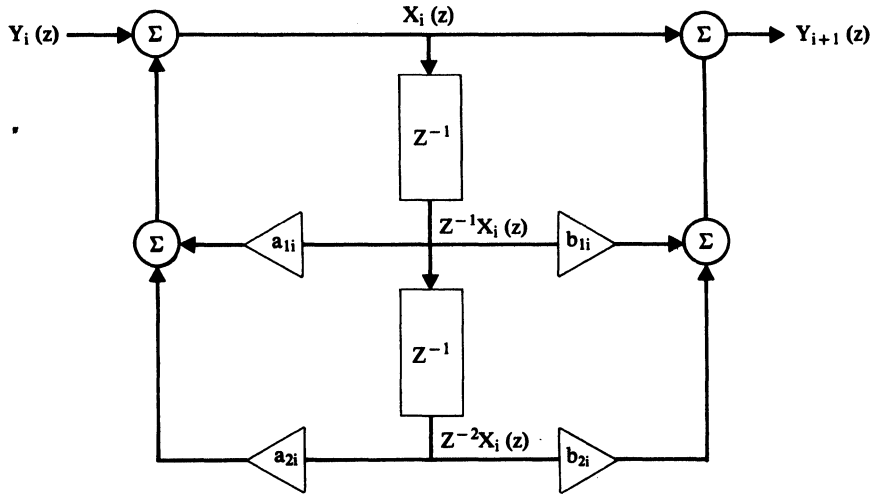


Fig. 7.3 Biquad Section Structure

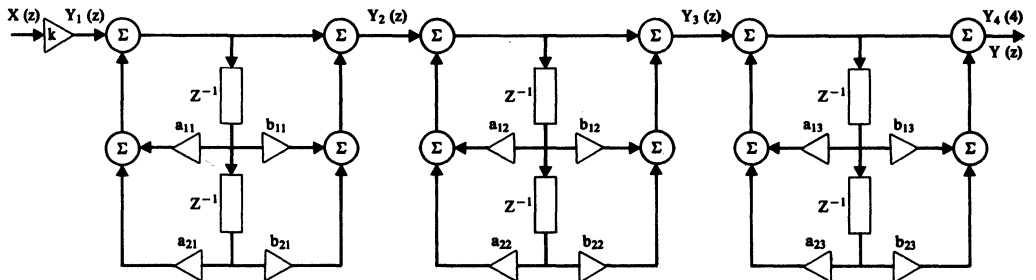


Fig. 7.4 Realization of a Six-order Digital Filter as a Cascade of Three Biquad Sections

$$s = \frac{2}{T} \frac{z-1}{z+1} \quad (7.10)$$

where T is the sampling period. To consider the frequency response, the Laplace variable s is evaluated on the imaginary axis and the z transform variable is evaluated on the unit circle:

$$s = j\Omega \quad (7.11)$$

$$z = e^{j\omega T} \quad (7.12)$$

where Ω is the analog frequency and ω is the digital frequency. The relationship between the analog and digital frequencies can be obtained using Equations 7.8, 7.9, and 7.10 as:

$$\begin{aligned}
\Omega &= \frac{2}{jT} \frac{e^{j\omega T} - 1}{e^{j\omega T} + 1} \\
&= \frac{2}{jT} \frac{e^{j\omega T/2}(e^{j\omega T/2} - e^{-j\omega T/2})}{e^{j\omega T/2}(e^{j\omega T/2} + e^{-j\omega T/2})} \\
&= \frac{2}{jT} \frac{2j\sin\omega T/2}{2\cos\omega T/2} \\
&= \frac{2}{T} \tan\omega T/2
\end{aligned} \tag{7.13}$$

The relationship between the analog frequency Ω and the digital frequency ω is shown in Fig. 7.5. It can be seen that the infinite imaginary axis in the s-plane ($-\infty \leq \Omega \leq \infty$) is mapped onto the unit circle in the Z plane ($-\pi \leq \omega T \leq \pi$). Furthermore, the frequencies Ω and ω have a nonlinear relationship that has a compression effect on the frequency-response characteristics. The compression effect is known as frequency warping. This compression causes the transfer function at the high Ω frequencies to be highly distorted when it is translated to the ω -domain. This compression can be alleviated by introducing a suitable prewarping to the Ω frequency scale to convert it to the Ω^* frequency scale, where

$$\Omega^* = \frac{2}{T} \tan\omega T/2 \tag{7.14}$$

In the following sections, the cutoff frequencies of the analog filters are first converted to the Ω^* frequency scale and then are used to design the analog filters.

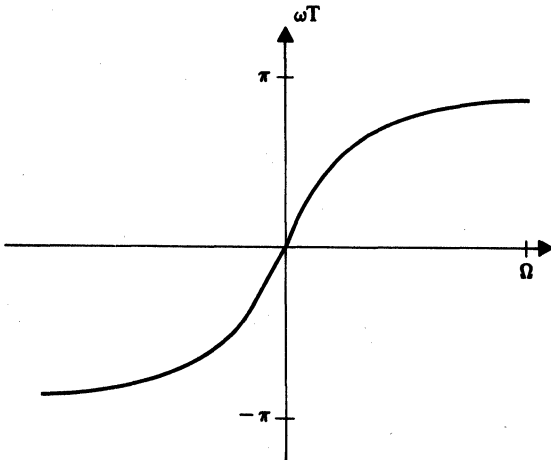


Fig. 7.5 Relationship Between Analog and Digital Frequencies

7.3 IIR FILTER SPECIFICATIONS

The frequency-domain specifications of standard IIR filters are the pass-band and stop-band attenuations, and the pass-band and stop-band cutoff frequencies. Figure 7.6 shows the frequency-domain specifications for the low-pass, high-pass, band-pass, and band-stop filters, where:

α = pass-band attenuation in dB

β = stop-band attenuation in dB

Ω_c = 3 dB cutoff frequency

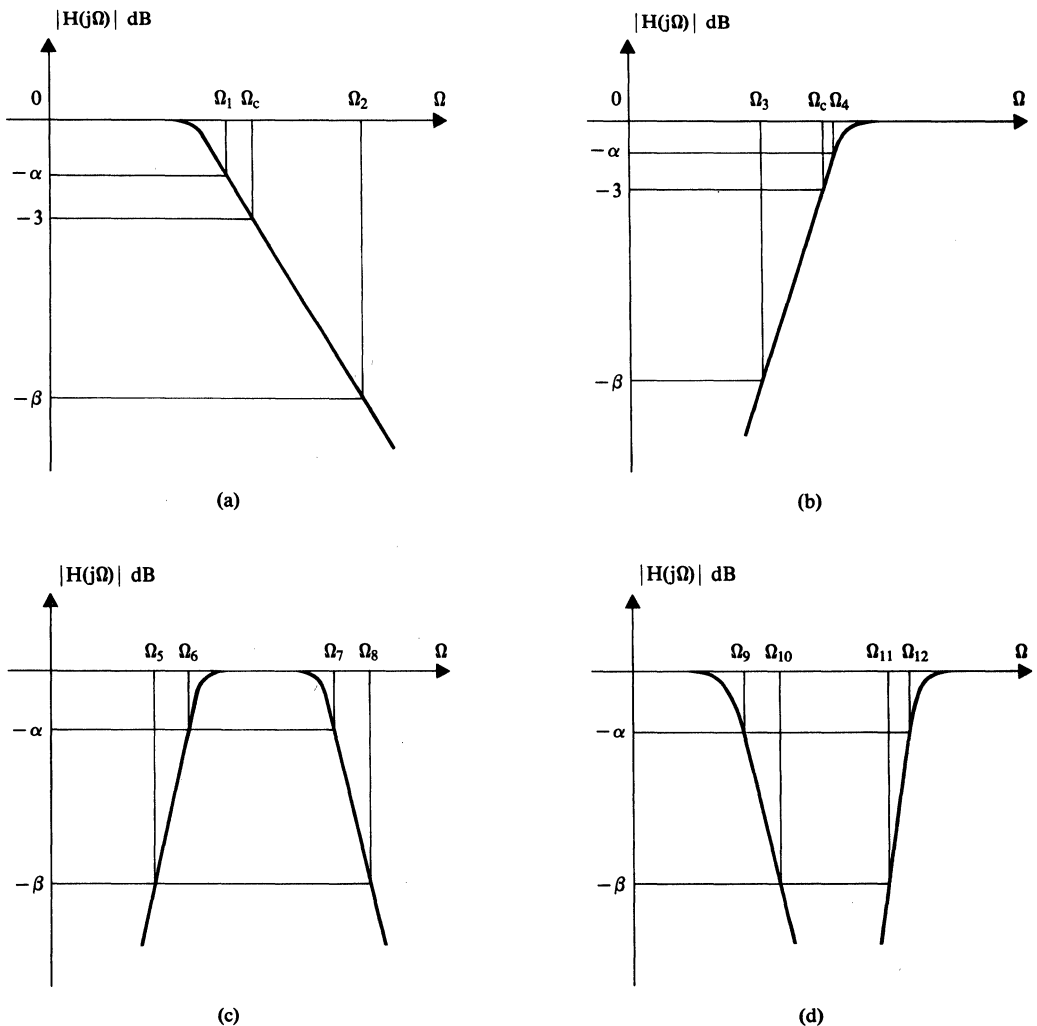


Fig. 7.6 Typical Analog Filters: a) Low-pass; b) High-pass; c) Band-pass; d) Band-stop

- Ω_1 = Pass-band cutoff frequency of low-pass filter
- Ω_2 = Stop-band cutoff frequency of low-pass filter
- Ω_3 = Stop-band cutoff frequency of high-pass filter
- Ω_4 = Pass-band cutoff frequency of high-pass filter
- Ω_5 = Stop-band lower cutoff frequency of band-pass filter
- Ω_6 = Pass-band lower cutoff frequency of band-pass filter
- Ω_7 = Pass-band upper cutoff frequency of band-pass filter
- Ω_8 = Stop-band upper cutoff frequency of band-pass filter
- Ω_9 = Pass-band lower cutoff frequency of band-stop filter
- Ω_{10} = Stop-band lower cutoff frequency of band-stop filter
- Ω_{11} = Stop-band upper cutoff frequency of band-stop filter
- Ω_{12} = Pass-band upper cutoff frequency of band-stop filter

The design process begins by prewarping the cutoff frequencies of the desired analog filter to obtain the corresponding cutoff frequencies on the Ω^* scale, where the relationship between Ω^* and Ω is shown in Equation 7.14. The transfer function of the analog filter is then designed to meet the specifications on the Ω^* scale. The digital filter transfer function obtained from the analog transfer function, by the Bilinear Transformation, will satisfy the specifications on the Ω scale. This can be seen by obtaining w from Equations 7.13 and 7.14 as:

$$w = \frac{2}{T} \arctan(\Omega^* T/2) = \Omega \quad (7.15)$$

7.4 NORMALIZED LOW-PASS FILTER DESIGN

In this section, the cutoff frequencies of the analog filter on the Ω^* scale are converted to corresponding cutoff frequencies of a normalized analog low-pass filter $H_n(s)$ by using an analog frequency transformation selected from Table 7.1. $H_n(s)$ is an approximation of the normalized ideal low-pass filter that has a gain of one in the band of frequencies from zero to one rad/sec and a gain of zero for all frequencies above one rad/sec.

For given normalized low-pass analog filter specifications, there are numerous types of filters that can be designed. The most popular are the Butterworth and Chebyshev filters. For $w \geq 0$, the Butterworth low-pass filter is characterized by a monotonically decreasing magnitude function of w , while the Chebyshev low-pass filter is characterized by an equiripple magnitude function across the passband and a monotonically decreasing magnitude function in the stopband. The normalized low-pass Butterworth and Chebyshev filters are shown in Fig. 7.7, where Ω_L^n and Ω_H^n are the normalized pass-band and stop-band cutoff frequencies, respectively.

The normalized pass-band cutoff frequency Ω_L^n is equal to one for the Chebyshev filter and is equal to the pass-band cutoff frequency divided by the 3 dB cutoff fre-

Table 7.1 Analog Frequency Transformations

Desired Filter	Analog Frequency Transformation
Low-pass	$s' = s/\Omega^+$
High-pass	$s' = \Omega^+/s$
Band-pass	$s' = \frac{s^2 + \Omega_7^* \Omega_6^*}{s(\Omega_7^* - \Omega_6^*)}$
Band-stop	$s' = \frac{s(\Omega_{11}^* - \Omega_{10}^*)\Omega_H^*}{s^2 + \Omega_{11}^* \Omega_{10}^*}$
where	$s' = j\Omega^n$ $s' = j\Omega^*$ $\Omega^+ = \Omega_c^*$ for Butterworth filter $= \Omega_L^*$ for Chebyshev filter $\Omega^n =$ normalized analog low-pass frequency

quency for the Butterworth filter. If the actual 3 dB cutoff frequency is not given, then a guess must be made to calculate Ω_L^n .

The ratio of Ω_H^n to Ω_L^n , referred to as R, can be calculated by using the analog frequency transformations and the low-pass, high-pass, band-pass, or band-stop cutoff frequencies as:

A. Low-pass to Normalized Low-pass:

From Table 7.1, the low-pass to normalized low-pass transformation is given by:

$$s' = s/\Omega^+ \text{ or } \Omega^n = \Omega^*/\Omega^+ \quad (7.16)$$

Hence, R can be calculated as:

$$\begin{aligned} R &= \Omega_H^n / \Omega_L^n \\ &= (\Omega_2^*/\Omega^+) / (\Omega_1^*/\Omega^+) = \Omega_2^* / \Omega_1^* \end{aligned} \quad (7.17)$$

B. High-pass to Normalized Low-pass:

From Table 7.1, the high-pass to normalized low-pass transformation is given by:

$$s' = \Omega^+/s \text{ or } \Omega^n = \Omega^+/\Omega^* \quad (7.18)$$

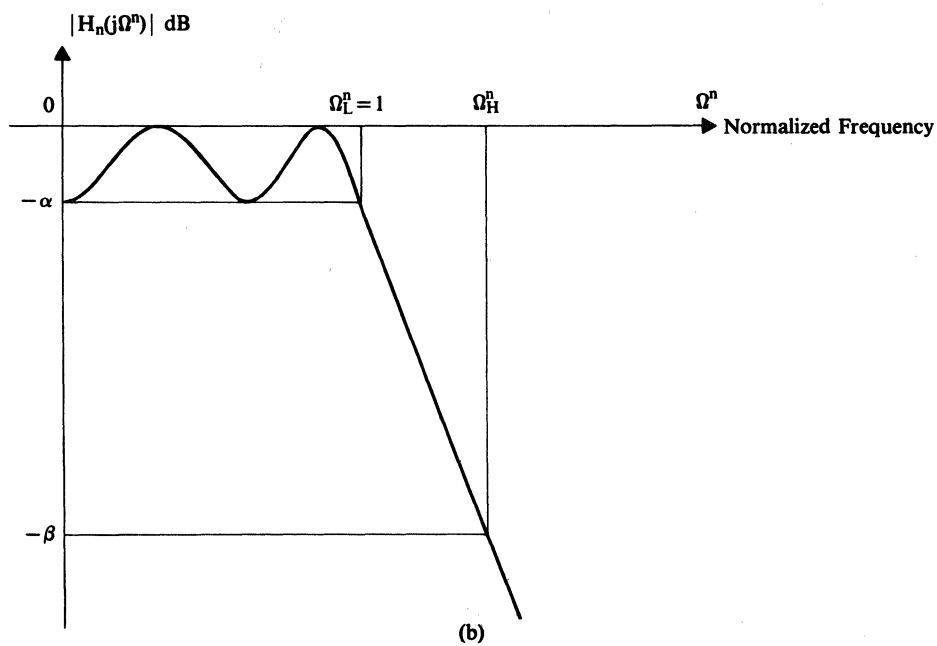
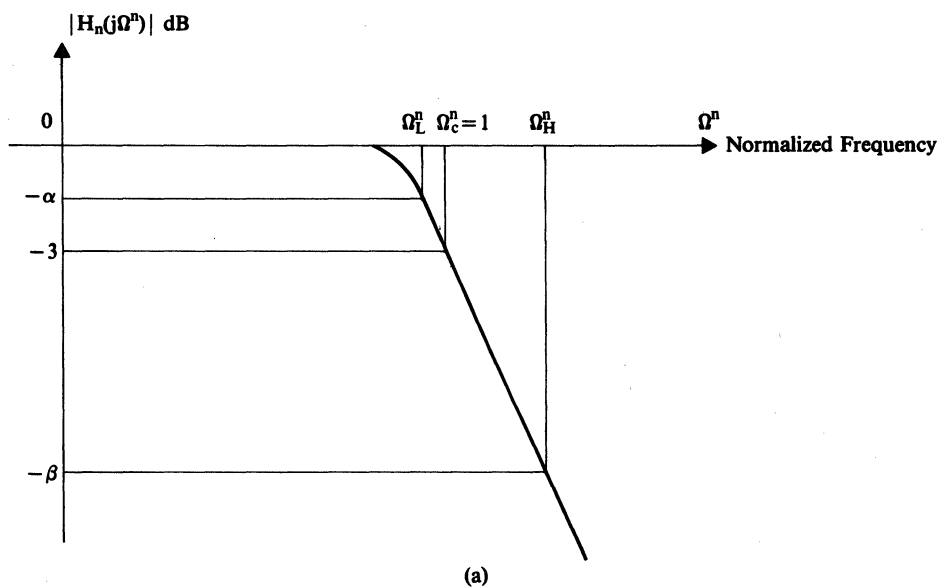


Fig. 7.7 Normalized Analog Low-pass Filter: a) Butterworth Filter; b) Chebyshev Filter

Using Equation 7.18, R can be calculated as:

$$\begin{aligned} R &= \Omega_H^n / \Omega_L^n \\ &= (\Omega^+ / \Omega_3^*) / (\Omega^+ / \Omega_4^*) = \Omega_4^* / \Omega_3^* \end{aligned} \quad (7.19)$$

C. Band-pass to Normalized Low-pass:

From Table 7.1, the band-pass to normalized low-pass transformation is given by:

$$s' = \frac{s^2 + \Omega_7^* \Omega_6^*}{s(\Omega_7^* - \Omega_6^*)} \quad \text{or} \quad \Omega^n = \frac{\Omega^{*2} - \Omega_7^* \Omega_6^*}{\Omega^* (\Omega_7^* - \Omega_6^*)} \quad (7.20)$$

The transformation will transform Ω_6^* and Ω_7^* into $\Omega_L^n = \pm 1$, and Ω_5^* and Ω_8^* into

$$\Omega_{H1}^n = \frac{\Omega_5^{*2} - \Omega_7^* \Omega_6^*}{\Omega_5^* (\Omega_7^* - \Omega_6^*)} \quad (7.21)$$

$$\Omega_{H2}^n = \frac{\Omega_8^{*2} - \Omega_7^* \Omega_6^*}{\Omega_8^* (\Omega_7^* - \Omega_6^*)} \quad (7.22)$$

In order to have a stop-band attenuation of at least β dB, the stop-band cutoff frequency Ω_H^n must be chosen as the smaller of the absolute value of Ω_{H1}^n and Ω_{H2}^n , respectively. Hence, R can be calculated as:

$$R = \min \left[\left| \frac{\Omega_5^{*2} - \Omega_7^* \Omega_6^*}{\Omega_5^* (\Omega_7^* - \Omega_6^*)} \right|, \left| \frac{\Omega_8^{*2} - \Omega_7^* \Omega_6^*}{\Omega_8^* (\Omega_7^* - \Omega_6^*)} \right| \right] \quad (7.23)$$

D. Band-stop to Normalized Low-pass:

The band-stop to normalized low-pass transformation is given by:

$$s' = \frac{s(\Omega_{11}^* - \Omega_{10}^*)\Omega_H^*}{s^2 + \Omega_{11}^* \Omega_{10}^*} \quad \text{or} \quad \Omega^n = \frac{\Omega^* (\Omega_{11}^* - \Omega_{10}^*)\Omega_H^*}{\Omega_{11}^* \Omega_{10}^* - \Omega^{*2}} \quad (7.24)$$

The transformation will transform Ω_{11}^* and Ω_{10}^* into $\Omega_L^n = \pm \Omega_H^n$, and Ω_9^* and Ω_{12}^* into:

$$\Omega_{L1}^n = \frac{\Omega_9^* (\Omega_{11}^* - \Omega_{10}^*) \Omega_H^n}{\Omega_{11}^* \Omega_{10}^* - \Omega_9^{*2}} \quad (7.25)$$

$$\Omega_{L2}^n = \frac{\Omega_{12}^* (\Omega_{11}^* - \Omega_{10}^*) \Omega_H^n}{\Omega_{11}^* \Omega_{10}^* - \Omega_{12}^{*2}} \quad (7.26)$$

In order to have a pass-band attenuation of at least α dB, the pass-band cutoff frequency Ω_L^n must be chosen as the larger of the absolute value of Ω_{L1}^n and Ω_{L2}^n , respectively.

Now R can be calculated as:

$$R = \min \left[\left| \frac{\Omega_{11}^* \Omega_{10}^* - \Omega_9^{*2}}{\Omega_9^* (\Omega_{11}^* - \Omega_{10}^*)} \right|, \left| \frac{\Omega_{11}^* \Omega_{10}^* - \Omega_{12}^{*2}}{\Omega_{12}^* (\Omega_{11}^* - \Omega_{10}^*)} \right| \right] \quad (7.27)$$

Table 7.2 shows the values of R that are calculated by using the low-pass, high-pass, band-pass, and band-stop cutoff frequencies. After calculating R , the normalized stop-band cutoff frequency Ω_H^n is obtained by multiplying Ω_L^n by R .

Table 7.2 Ratio of Stop-band to Pass-band Cutoff Frequencies

Desired Filter	R for Normalized Low-pass Filter $R = \Omega_H^n / \Omega_L^n$
Low-pass	Ω_2^* / Ω_1^*
High-pass	Ω_4^* / Ω_3^*
Band-pass	$\min \left[\left \frac{\Omega_5^{*2} - \Omega_6^* \Omega_7^*}{\Omega_5^* (\Omega_7^* - \Omega_6^*)} \right , \left \frac{\Omega_8^{*2} - \Omega_6^* \Omega_7^*}{\Omega_8^* (\Omega_7^* - \Omega_6^*)} \right \right]$
Band-Stop	$\min \left[\left \frac{\Omega_{11}^* \Omega_{10}^* - \Omega_9^{*2}}{\Omega_9^* (\Omega_{11}^* - \Omega_{10}^*)} \right , \left \frac{\Omega_{11}^* \Omega_{10}^* - \Omega_{12}^{*2}}{\Omega_{12}^* (\Omega_{11}^* - \Omega_{10}^*)} \right \right]$

7.4.1 Butterworth Filter Design

The amplitude response of an N^{th} -order normalized low-pass analog Butterworth filter can be written as:

$$|H_n(j\Omega^n)|^2 = \frac{1}{1 + (\Omega^n)^{2N}} \quad (7.28)$$

The first step in the design process is to find the order of the filter N that meets the normalized low-pass filter specifications (α , β , Ω_L^n , and Ω_H^n). This can be accomplished by choosing N as the smallest even integer satisfying:

$$N \geq \max(N_1, N_2) \quad (7.29)$$

where N_1 and N_2 satisfy the following inequalities:

$$-10 \log \frac{1}{1 + (\Omega_L^n)^{2N_1}} < \alpha \quad (7.30)$$

$$-10 \log \frac{1}{1 + (\Omega_H^n)^{2N_2}} > \beta \quad (7.31)$$

The next step in the design process is to find the normalized transfer function $H_n(s)$. Substituting $s' = j\Omega^n$ into Equation 7.28 yields:

$$|H_n(s')|^2 = \frac{1}{1 + (-1)^N s'^{2N}} \quad (7.32)$$

The denominator has the following $2N$ roots:

$$s'_i = \exp j\pi(2i + N - 1)/2N \quad 1 \leq i \leq 2N \quad (7.33)$$

The $2N$ roots of are located on a unit circle in the s' -plane as shown in Fig. 7.8. To form a stable N th order normalized low-pass filter, the poles in the left half of the s' -plane are used. The related transfer function is:

$$H_n(s') = \frac{1}{(s' - s'_1)(s' - s'_2) \dots (s' - s'_{N'})} \quad (7.34)$$

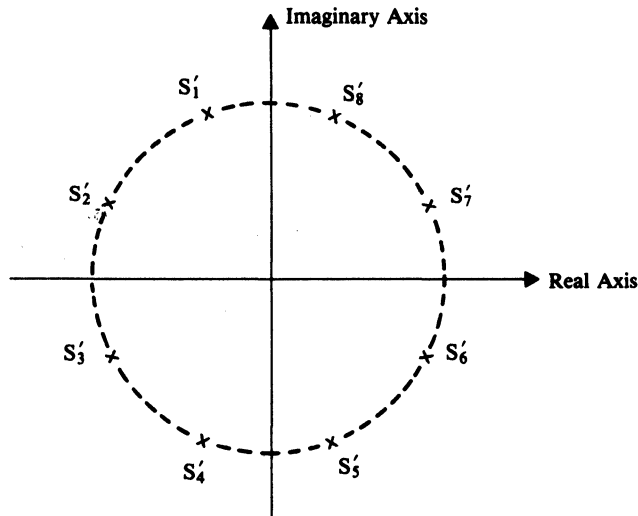


Fig. 7.8 Butterworth Poles for $N = 4$

For N even, all poles occur in complex conjugate pairs of the form s'_i, s'_{N+1-i} , where $1 \leq i \leq N/2$. Using the pole definitions in Equation 7.33, the transfer function for a normalized filter can be rewritten as:

$$H_n(s') = \prod_{i=1}^{M=0.5N} H_i(s') \quad (7.35)$$

where

$$\begin{aligned} H_i(s') &= \frac{1}{(s' - s'_i)(s' - s'_{N+1-i})} \\ &= \frac{1}{s'^2 - 2s' \cos((2i + N - 1)\pi/2N) + 1} \end{aligned} \quad (7.36)$$

7.4.2 Chebyshev Filter Design

The magnitude response of an N^{th} -order Chebyshev filter can be written as:

$$|H_n(j\Omega^n)|^2 = \frac{1}{1 + \epsilon^2 T_N^2(\Omega^n)} \quad (7.37)$$

where $T_N(\Omega^n)$ is the N^{th} -order Chebyshev polynomial:

$$\begin{aligned} T_N(\Omega^n) &= \cos(N \cos^{-1} \Omega^n) & \Omega^n \leq 1 \\ &= \cosh(N \cosh^{-1} \Omega^n) & \Omega^n > 1 \end{aligned} \quad (7.38)$$

and ϵ is a parameter that controls the passband ripple. The passband amplitude oscillates between 1 and $1/(1+\epsilon^2)^{0.5}$. The amplitude at $\Omega^n = 0$ is one if N is odd and $1/(1+\epsilon^2)^{0.5}$ if N is even. The amplitude at the normalized cutoff frequency of 1 rad/sec is equal to $1/(1+\epsilon^2)^{0.5}$ and, consequently, the passband attenuation α can be written as:

$$-10 \log \frac{1}{(1 + \epsilon^2)} = \alpha \quad (7.39)$$

Hence, the ripple parameter ϵ is determined as:

$$\epsilon = (10^{0.1\alpha} - 1)^{0.5} \quad (7.40)$$

Similarly, the stopband attenuation β can be written, by using Equation 7.37, as:

$$-10 \log \frac{1}{(1 + \epsilon^2 T_N^2(\Omega_H^n))} = \beta \quad (7.41)$$

Using Equations 7.38 and 7.41, the Chebyshev polynomial $T_N(\Omega_H^n)$ at the normalized stopband cutoff frequency Ω_H^n can be written as:

$$\begin{aligned} T_N(\Omega_H^n) &= [(10^{0.1\beta} - 1)/\epsilon]^{0.5} \\ &= \cosh(N \cosh^{-1} \Omega_H^n) \end{aligned} \quad (7.42)$$

From Equation 7.42, the order of the Chebyshev filter N is the smallest integer satisfying:

$$N \geq \frac{\cosh^{-1} [(10^{0.1\beta} - 1)/\epsilon]^{0.5}}{\cosh^{-1} \Omega_H^n} \quad (7.43)$$

As in the previously designed Butterworth filter, the normalized Chebyshev filter is specified in terms of the pole locations on the s' -plane. The poles of this filter lie on an ellipse in the s' -plane as shown in Fig. 7.9. To form a stable normalized low-pass filter, the poles that lie in the stable region are selected. For even N , the stable poles s_i' are given by:

$$s_i' = \mu_i + j\sigma_i \quad (7.44)$$

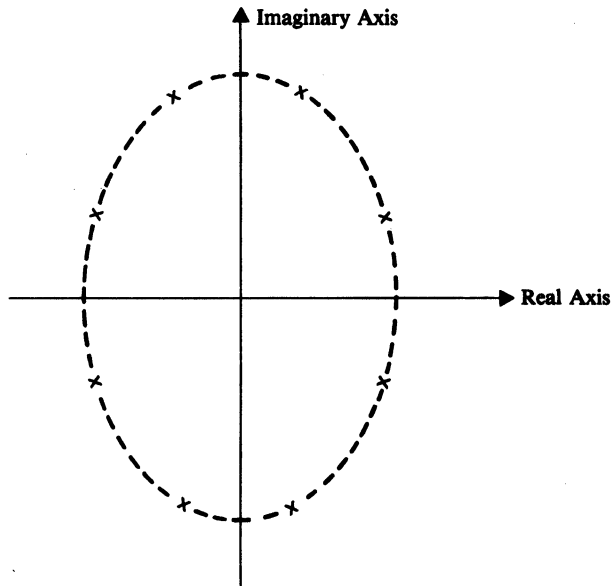


Fig. 7.9 Chebyshev Poles for $N = 4$

where

$$\mu_i = -\sinh \gamma \cos \frac{i\pi}{2N} \quad i = \pm 1, \pm 3, \pm 5, \dots, \pm N-1 \quad (7.45)$$

$$\sigma_i = \cosh \gamma \sin \frac{i\pi}{2N} \quad i = \pm 1, \pm 3, \pm 5, \dots, \pm N-1 \quad (7.46)$$

$$\gamma = \frac{1}{N} \sinh^{-1} \frac{1}{\varepsilon} \quad (7.47)$$

As in the previous section, the complex-conjugate roots can be combined to be the roots of a second-order polynomial. For N even, $H_n(s')$ has the form:

$$H_n(s') = \prod_i \frac{1}{s'^2 - 2\mu_i s' + (\mu_i^2 + \sigma_i^2)} \quad (7.48)$$

where $i = 1, 3, 5, \dots, N-1$

7.5 ANALOG FILTER DESIGN

The analog frequency transformations are used to convert the normalized low-pass analog filter to the desired analog filter, where the transfer function of the analog low-pass, high-pass, band-pass, and band-stop filter can be written as:

$$H_{LP}(s) = H_n(s') \downarrow_{s' = s/\Omega^+} \quad (7.49)$$

$$H_{HP}(s) = H_n(s') \downarrow_{s' = \Omega^+/s} \quad (7.50)$$

$$H_{BP}(s) = H_n(s') \downarrow_{s' = \frac{s^2 + \Omega_7^* \Omega_6^*}{s(\Omega_7^* - \Omega_6^*)}} \quad (7.51)$$

$$H_{BS}(s) = H_n(s') \downarrow_{s' = \frac{s(\Omega_{11}^* - \Omega_{10}^*)\Omega_H^*}{s^2 + \Omega_{11}^* \Omega_{10}^*}} \quad (7.52)$$

7.6 DIGITAL FILTER DESIGN

The final step in the design process is to obtain the desired digital filter from the analog filter by using the Bilinear Transformation:

$$H(z) = H(s) \downarrow_{s = (2/T)(z - 1/z + 1)} \quad (7.53)$$

The transfer function $H(z)$ can be decomposed into a product of biquad sections. The biquad sections are described by Equations 7.3 through 7.6. In the following chapter sections, the coefficients of the biquad sections b_{1i} , b_{2i} , a_{1i} , and a_{2i} and the total gain K are used to implement the digital IIR filter on the DSP56002 processor.

7.7 IMPLEMENTING A BIQUAD SECTION ON THE DSP56002 PROCESSOR

For the i^{th} biquad section in Figure 7.3, the input sample $y_i(n)$ is in Accumulator A, and the output sample $y_{i+1}(n)$ will be in Accumulator A. This allows $y_{i+1}(n)$ to be the input sample for the $(i+1)^{\text{th}}$ section.

From Equations 7.8 and 7.9, the difference equations for the i^{th} biquad section can be written as:

$$x_i(n) = y_i(n) + a_{1i}x_i(n-1) + a_{2i}x_i(n-2) \quad (7.54)$$

$$y_{i+1}(n) = x_i(n) + b_{1i}x_i(n-1) + b_{2i}x_i(n-2) \quad (7.55)$$

Since the actual biquad coefficients vary from -2 to +2, the scaling mode is used, and the coefficients a_{1i} , a_{2i} , b_{1i} , b_{2i} are the actual values for the filter divided by two.

The X-Memory map for the filter states and the Y-Memory map for the coefficients before the execution of the i^{th} biquad section are shown in Fig. 7.10. R2 is initialized to point to the filter states in X-Memory, and R4 is initialized to point to the filter coefficients in Y-Memory. Note the sequence in which the filter states are stored. The first element of the sequence is the second filter state $x_i(n-2)$, and the second element is the first filter state $x_i(n-1)$. Similarly, the first element of the coefficients is a_{2i} then a_{1i} , followed by b_{2i} then b_{1i} . The initial values in R2 and R4 are assumed to be \$100.

The following DSP56002 instructions are used to implement the i^{th} biquad section:

```

ori      #$8, mr
move     x0, y0, a      x: (r2)+, x0      y: (r4)+, y0
mac      x0, y0, a      x: (r2)-, x1      y: (r4)+, y0
macr     x1, y0, a      x1, x: (r2)+      y: (r4)+, y0
mac      x0, y0, a      a, x: (r2)+      y: (r4)+, y0
mac      x1, y0, a      x: (r2)+, x0      y: (r4)+, y0

```

The i^{th} biquad section operates as follows:

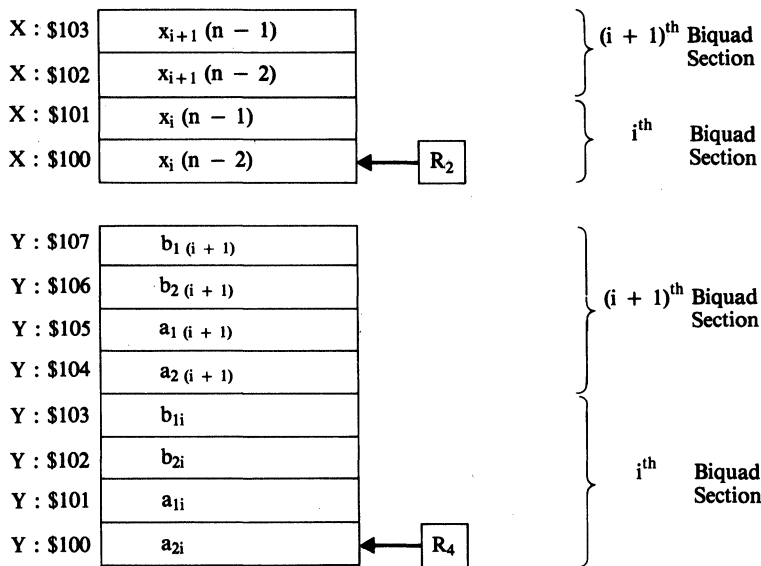


Fig. 7.10 Memory Map Before Execution of the i^{th} Biquad Section

1. The **ORI** instruction selects the DSP56002 processor's up-scaling mode.
2. The **MOVE** instruction transfers $x_i(n-2)$ from the X-Memory location \$100, pointed to by R_2 , to the Data ALU's Input Register X0 and transfers a_{2i} from the Y-Memory location \$100, pointed to by R_4 , to the Data ALU's Input Register Y0. Address Register R_2 is postincremented by one to point to $x_i(n-1)$ in X-Memory location \$101; R_4 is postincremented by one to point to a_{1i} in Y-Memory location \$101. The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A-Accumulator and Data ALU Input Registers X0, X1, and Y0 after the execution of the **MOVE** instruction are as shown in Fig. 7.11.
3. The first **MAC** instruction multiplies $x_i(n-2)$ from X0 by a_{2i} from Y0, and adds the product to $y_i(n)$ from the A-Accumulator; i.e., $[x_i(n-2) a_{2i}] + y_i(n) \rightarrow \text{A-Accumulator}$. In the parallel-move portion of the **MAC** instruction, $x_i(n-1)$ is transferred from X-Memory location \$101, pointed to by R_2 , to X1, and the next coefficient a_{1i} is transferred from Y-Memory location \$101, pointed to by R_4 , to Y0. Also, R_2 is postdecremented by one to point back to $x_i(n-2)$ in X-Memory location \$100, and R_4 is incremented by one to point to b_{2i} in Y-Memory location \$102. The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A-Accumulator and Data ALU Input Registers X0, X1, and Y0 after the execution of the first **MAC** instruction are as shown in Fig. 7.12.
4. The **MACR** instruction multiplies $x_i(n-1)$ from X1 by a_{1i} from Y0, adds the product to $[y_i(n) + a_{2i}x_i(n-2)]$ from the A-Accumulator, and convergently rounds the result; i.e., $x_i(n-1) a_{1i} + [y_i(n) + a_{2i}x_i(n-2)] \rightarrow \text{A-Accumulator}$ with convergent

rounding of the result. In the parallel-move portion of the **MACR** instruction, $\mathbf{x}_i(\mathbf{n}-1)$ from X1 is transferred to X-Memory location \$100, pointed to by R2, and the next coefficient $\mathbf{b}_{2i}$ is transferred from Y-Memory location \$102, pointed to by R4, to Y0. Also, R2 is postincremented by one to point to $\mathbf{x}_i(\mathbf{n}-1)$ in X-Memory location \$101; R4 is postincremented by one to point to $\mathbf{b}_{1i}$ in Y-Memory location \$103. Note that $\mathbf{x}_i(\mathbf{n}-1)$ overwrites $\mathbf{x}_i(\mathbf{n}-2)$ at X-Memory location \$100. The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A-Accumulator and Data ALU Input Registers X0, X1, and Y0 after the execution of the **MACR** instruction are as shown in Fig. 7.13.

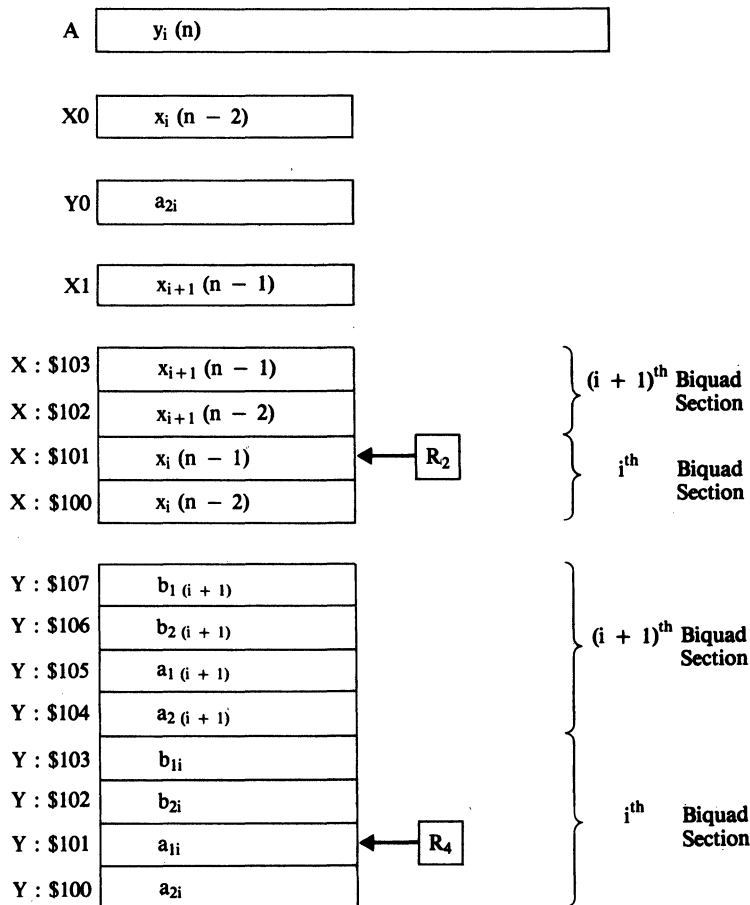


Fig. 7.11 Memory Map and Data Registers After Execution of the Move Instruction

5. The second **MAC** instruction multiplies $x_i(n-2)$ from X0 by b_{2i} from Y0, and adds the product to the contents of the A-Accumulator. In the parallel-move portion of this **MAC** instruction, $x_i(n)$ from the A-Accumulator is up-scaled by two and transferred to X-Memory location \$101, pointed to by R2, and the next coefficient b_{1i} is transferred from Y-Memory location \$103, pointed to by R4, to Y0. Also, R2 is postincremented by one to point to $x_{i+1}(n-2)$ of the $i+1^{\text{th}}$ biquad filter section in X-Memory location \$102; R4 is postincremented by one to point to coefficient

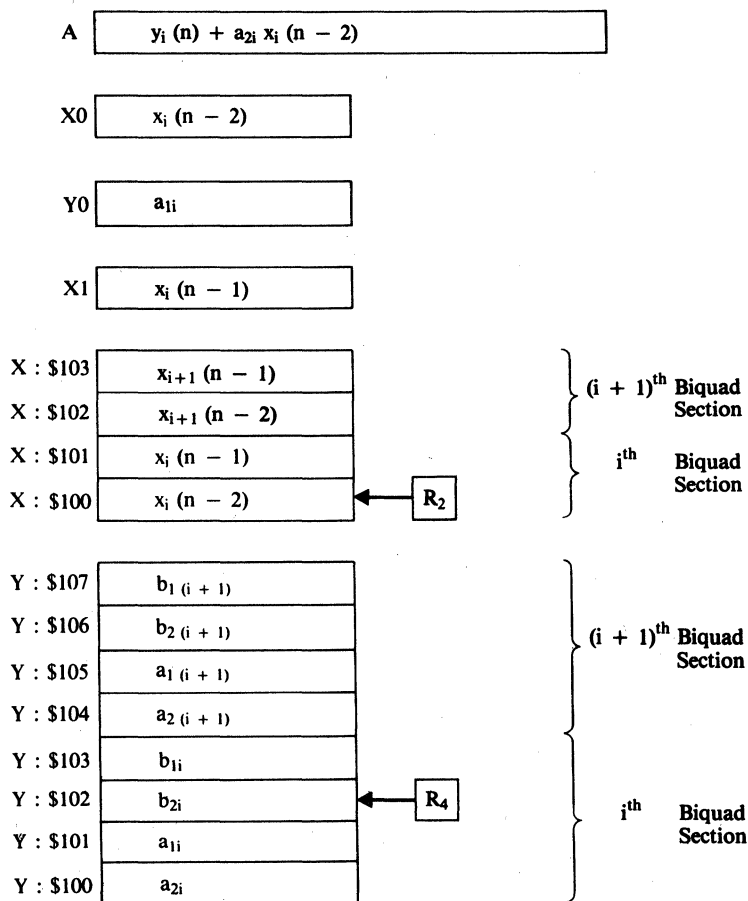


Fig. 7.12 Memory Map and Data Registers After Execution of the First MAC Instruction

$a_{2(i+1)}$ of the $i+1^{\text{th}}$ biquad filter section in Y-Memory location \$104. Note that $x_i(n)$, up-scaled by two, overwrites $x_i(n-1)$ at X-Memory location \$101. The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A-Accumulator and Data ALU Input Registers X0, X1, and Y0 after the execution of the second **MAC** instruction are as shown in Fig. 7.14.

6. The last **MAC** instruction multiplies $x_i(n-1)$ from X1 by b_{1i} from Y0, and adds the product to the contents of the A-Accumulator. The result in the A-Accumula-

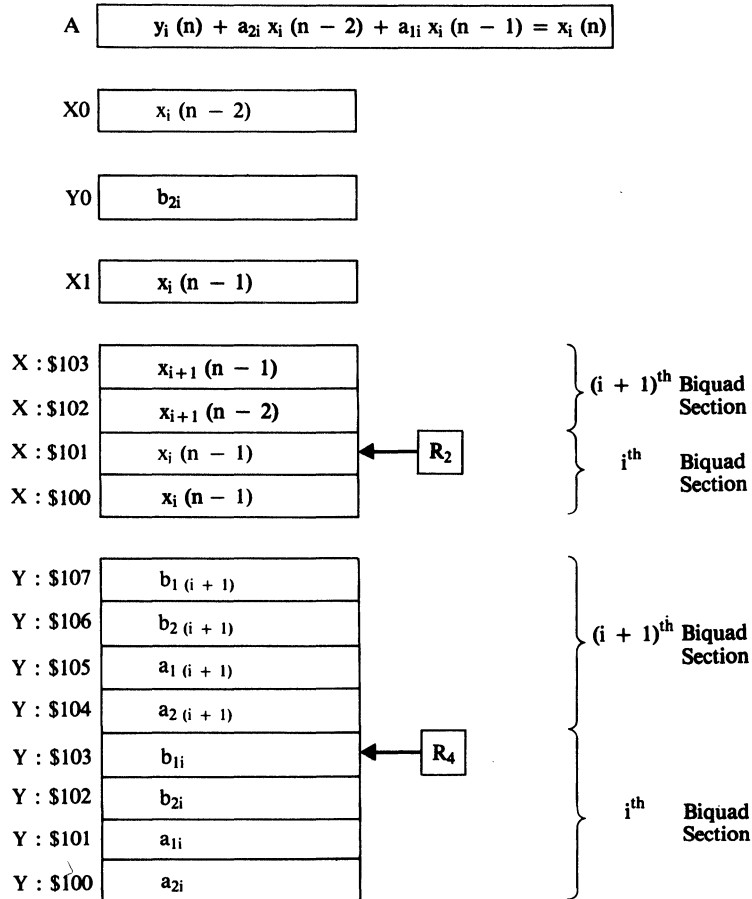


Fig. 7.13 Memory Map and Data Registers After Execution of the MACR Instruction

tor, then, is the output of the biquad section $y_{i+1}(n)$ and the input to the $i+1^{\text{th}}$ biquad section. In the parallel-move portion of this **MAC** instruction, $x_{i+1}(n-2)$ is transferred from X-Memory location \$102, pointed to by R2, to X0; and the first coefficient $a_{2(i+1)}$ of the $i+1^{\text{th}}$ biquad section is transferred from Y-Memory location \$104, pointed to by R4, to Y0. Also, R2 is postincremented by one to point to $x_{i+1}(n-1)$ in X-Memory location \$103; R4 is postincremented by one to point to

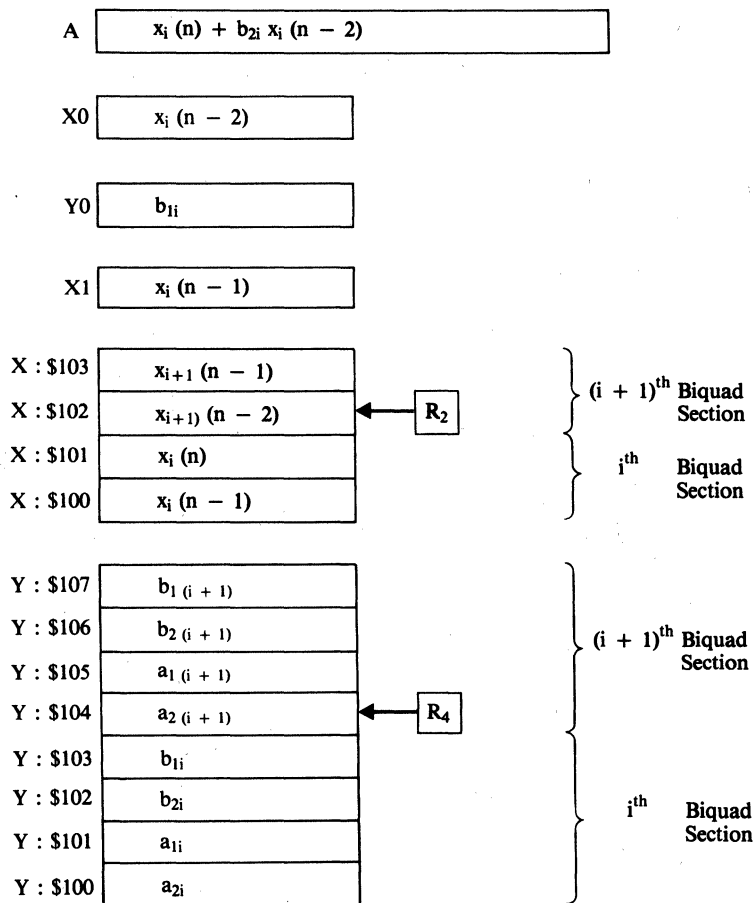


Fig. 7.14 Memory Map and Data Registers After Execution of the Second MAC Instruction

$a_{1(i+1)}$ in Y-Memory location \$105. Note that the conditions at the beginning of the $i+1^{\text{th}}$ biquad section are exactly the same as the conditions that existed at the beginning of the i^{th} biquad section. The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A-Accumulator and Data ALU Input Registers X0, X1, and Y0 after the execution of the last **MAC** instruction are as shown in Fig. 7.15.

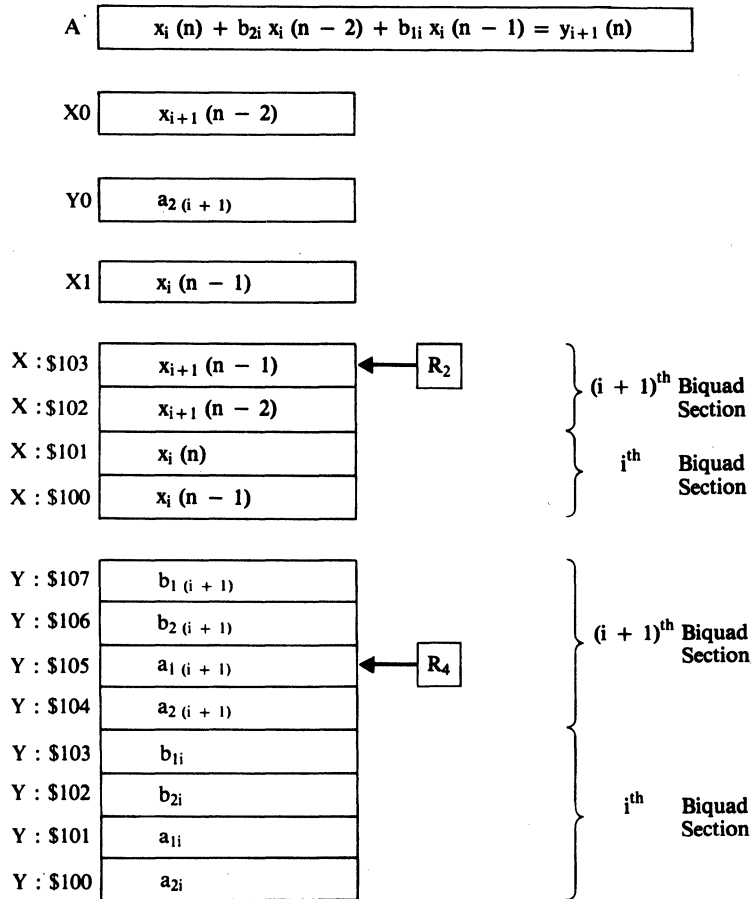


Fig. 7.15 Memory Map and Data Registers After Execution of the Last MAC Instruction

7.8 IMPLEMENTING AN IIR FILTER

A desired IIR filter is implemented by cascading biquad sections as shown in Figure 7.4. The biquad sections themselves are implemented as described above.

The DSP56002 instructions that generate the input $y_1(n)$ to the first biquad section are shown in Fig. 7.16. Initially, the first element of the input sequence $x(n)$ is in the A-Accumulator. The first **MOVE** instruction transfers the 8-bit immediate pointer value **gk** to R1. Since R1 has just been changed, the first **NOP** instruction is used to provide sufficient access time such that the X-Memory data pointed to by R1 is available for use during the second **MOVE** instruction. The second **MOVE** instruction transfers the “**total gain**” value, from the X-Memory location pointed to by R1, to Y1. The third **MOVE** instruction transfers $x(n)$ from the A-Accumulator to X1. In the **MPY** instruction, $x(n)$ from X1 is multiplied by **total gain** from Y1 to place the product $y_1(n)$ into the A-Accumulator.

Fig. 7.16 DSP56002 Assembler Instructions to Generate the $y_1(n)$ Input to the First Biquad Section

```
move    #gk,r1
nop
move    x:(r1),y1
move    a,x1
mpy     x1,x0,a
```

The DSP56002 instructions that implement **NSEC** cascaded biquad sections, where **NSEC** specifies the quantity of the biquad sections, are shown in Fig. 7.17. Note, the core biquad section instructions were explained in section 7.7. Here, the **DO** instruction sets up a hardware “do loop” to execute, **NSEC** times, the instruction group that implements a biquad section. The **RND** instruction convergently rounds the contents of the A-Accumulator to produce the final 24-bit output of the IIR filter. The **ANDI** instruction disables the up-scaling mode.

Fig. 7.17 DSP56002 Assembler Instructions to Implement Cascaded Biquad Sections

```
ori     #$8,mr
nop
move    x:(r2)+,x0    y:(r4)+,y0

do      #NSEC,_iirend
mac     x0,y0,a        x:(r2)-,x1    y:(r4)+,y0
macr    x1,y0,a        x1,x:(r2)+    y:(r4)+,y0
mac     x0,y0,a        a,x:(r2)+    y:(r4)+,y0
mac     x1,y0,a        x:(r2)+,x0    y:(r4)+,y0
_iirend

rnd     a
andi    #$f7,mr
```

7.9 DIGITAL FILTER DESIGN AND ANALYSIS SYSTEM PACKAGE

The Digital Filter Design and Analysis System (QEDesign 1000 for Windows) software program by Momentum Data Systems Inc. is used to design filters and graphically show the filters' response in both the frequency and time domains. For more information about the QEDesign 1000 software program, please phone Momentum Data Systems Inc., Costa Mesa, California, at (714) 557-6884.

7.10 DSP56002 MACRO FOR IMPLEMENTING AN IIR FILTER ON THE DSP SYSTEM

The STIIR.ASM Assembler Macro shown in Fig. 7.18 processes the left channel analog input signal and generates two analog output signals. The left channel analog output signal is the filtered version of the left channel analog input signal. The right channel analog output signal is the left channel analog input signal. The filter of the left channel is implemented using DSP56002 instructions shown in Figure 7.17.

Fig. 7.18 STIIR.ASM Assembler Macro

```
stiir      macro    states,coef,gk,nsec
;*****
; Right Channel
;*****
        move        x0,x1
        move        x0,b
;*****
; Left Channel
;*****
        move        #gk,r1
        nop
;
;*****
; Multiply the input by the overall gain
;*****
;
        move        x:(r1),x0
        nop
        mpy         x1,x0,a
        nop
;
;*****
; Initialize R2, R4, M2 and M4
;*****
;
        move        #$ffff,m2
```

```

        move        m2,m4
        move        #states,r2
        move        #coef,r4
;
;*****
;  IIR Filter
;*****
;
        ori        #$8,mr
        nop
        move                x:(r2)+,x0        y:(r4)+,y0
;
        do          #nsec,_iirend
        mac         x0,y0,a        x:(r2)-,x1        y:(r4)+,y0
        macr        x1,y0,a        x1,x:(r2)+        y:(r4)+,y0
        mac         x0,y0,a        a,x:(r2)+        y:(r4)+,y0
        mac         x1,y0,a        x:(r2)+,x0        y:(r4)+,y0
_iirend
;
        rnd         a
        andi        #$f7,mr
        rol         a

        endm

```

7.11 IIR FILTER DEMONSTRATION SYSTEM

The IIR filtering process can be demonstrated by using the DSP system described in chapter 5 and shown in Fig. 5.1. The DSP system consists of two analog-to-digital (A/D) converters, the DSP56002 processor, and two digital-to-analog (D/A) converters. The DSP56002EVM Evaluation Module (EVM) is used to provide and control the DSP56002 processor, the two A/D converters, and the two D/A converters. The left channel analog input signal $x(t)$ is first digitized using the A/D converter on the EVM. The DSP56002 processor executes the IIR filter algorithm to process the left channel digitized input signal $x(n)$ and generate the left and right channel digital output signals $y_1(n)$ and $y_2(n)$. The left channel digital output signal $y_1(n)$ is the filtered version of the left channel digitized input signal $x(n)$. The right channel digital output signal $y_2(n)$ is the left channel digitized input signal $x(n)$. The two D/A converters on the EVM are then used to convert the left and right channel digital output signals $y_1(n)$ and $y_2(n)$ to the left and right channel analog output signals $y_1(t)$ and $y_2(t)$.

7.12 DESIGNING AND IMPLEMENTING IIR FILTERS

The **QEDesign 1000 for Windows** program will be used in sections 7.12.1 through 7.12.4 to design low-pass, high-pass, band-pass, and band-stop IIR filters, respectively.

The designed filter coefficients, the STIIR.ASM assembler program shown in Fig. 7.18, and the Init_system assembler program shown in Fig. 5.15 will be used to implement each IIR filter on the DSP system.

7.12.1 Designing and Implementing a Low-Pass IIR Filter

The following low-pass IIR filter is designed using QEDesign 1000 for Windows:

7.12.1.1 Filter Specifications:

Filter Type:	Low-pass
Analog Filter Type:	Butterworth
Sampling Frequency:	48000 Hz
Passband Cutoff Frequency:	8000 Hz
Stopband Cutoff Frequency:	10000 Hz
Passband attenuation:	0.5 dB
Stopband Attenuation:	30.0 dB
Filter Design Method:	Bilinear Transformation

7.12.1.2 Design Results

FILTER COEFFICIENT FILE
IIR DESIGN

```

FILTER TYPE                LOW PASS
ANALOG FILTER TYPE        BUTTERWORTH
PASSBAND RIPPLE IN -dB     -.5000
STOPBAND RIPPLE IN -dB     -30.0000
PASSBAND CUTOFF FREQUENCY  .800000E+04 HERTZ
STOPBAND CUTOFF FREQUENCY  .100000E+05 HERTZ
SAMPLING FREQUENCY        .480000E+05 HERTZ
FILTER DESIGN METHOD:      BILINEAR TRANSFORMATION
FILTER ORDER:             16    0010h
NUMBER OF SECTIONS:       8     0008h
NO. OF QUANTIZED BITS:    24    0018h
QUANTIZATION TYPE:        FRACTIONAL FIXED POINT

```

COEFFICIENTS SCALED FOR CASCADE FORM II

```

      0    00000000    /* shift count for overall gain    */
4892391  004AA6E7    /* overall gain      */
      0    00000000    /* shift count for section 1 values */
1245628  001301BC    /* section 1 coefficient B0      */
2491256  00260378    /* section 1 coefficient B1      */
1245628  001301BC    /* section 1 coefficient B2      */

```

```

3988293 003CDB45 /* section 1 coefficient A1 */
-492076 FFF87DD4 /* section 1 coefficient A2 */
0 00000000 /* shift count for section 2 values */
1292324 0013B824 /* section 2 coefficient B0 */
2584649 00277049 /* section 2 coefficient B1 */
1292324 0013B824 /* section 2 coefficient B2 */
4061760 003DFA40 /* section 2 coefficient A1 */
-655666 FFF5FECE /* section 2 coefficient A2 */
0 00000000 /* shift count for section 3 values */
1366707 0014DAB3 /* section 3 coefficient B0 */
2733414 0029B566 /* section 3 coefficient B1 */
1366707 0014DAB3 /* section 3 coefficient B2 */
4214028 00404D0C /* section 3 coefficient A1 */
-994719 FFF0D261 /* section 3 coefficient A2 */
0 00000000 /* shift count for section 4 values */
1474740 001680B4 /* section 4 coefficient B0 */
2949481 002D0169 /* section 4 coefficient B1 */
1474740 001680B4 /* section 4 coefficient B2 */
4456575 0044007F /* section 4 coefficient A1 */
-1534795 FFE894B5 /* section 4 coefficient A2 */
0 00000000 /* shift count for section 5 values */
1625863 0018CF07 /* section 5 coefficient B0 */
3251727 00319E0F /* section 5 coefficient B1 */
1625863 0018CF07 /* section 5 coefficient B2 */
4808853 00496095 /* section 5 coefficient A1 */
-2319208 FFD0C9C8 /* section 5 coefficient A2 */
0 00000000 /* shift count for section 6 values */
1834776 001BFF18 /* section 6 coefficient B0 */
3669553 0037FE31 /* section 6 coefficient B1 */
1834776 001BFF18 /* section 6 coefficient B2 */
5301636 0050E584 /* section 6 coefficient A1 */
-3416482 FFCBDE5E /* section 6 coefficient A2 */
0 00000000 /* shift count for section 7 values */
1651861 00193495 /* section 7 coefficient B0 */
3303722 0032692A /* section 7 coefficient B1 */
1651861 00193495 /* section 7 coefficient B2 */
5982863 005B4A8F /* section 7 coefficient A1 */
-4933363 FFB4B90D /* section 7 coefficient A2 */
0 00000000 /* shift count for section 8 values */
2697311 0029285F /* section 8 coefficient B0 */
5394622 005250BE /* section 8 coefficient B1 */
2697311 0029285F /* section 8 coefficient B2 */
6927884 0069B60C /* section 8 coefficient A1 */
-7037630 FF949D42 /* section 8 coefficient A2 */

.1484904289245605D+00 3FC301BC00000000 .14849047E+00 /* section 1 B0 */
.2969808578491211D+00 3FD301BC00000000 .29698093E+00 /* section 1 B1 */
.1484904289245605D+00 3FC301BC00000000 .14849047E+00 /* section 1 B2 */
.475441575053540D+00 3FDE6DA280000000 .47544161E+00 /* section 1 A1 */
-.5866003036499023D-01 BFAE08B000000000 -.58660148E-01 /* section 1 A2 */
.1540570259094238D+00 3FC3B82400000000 .15405711E+00 /* section 2 B0 */

```

```

.3081141710281372D+00 3FD3B82480000000 .30811422E+00 /* section 2 B1 */
.1540570259094238D+00 3FC3B82400000000 .15405711E+00 /* section 2 B2 */
.4841995239257812D+00 3FDEFD2000000000 .48419961E+00 /* section 2 A1 */
-.7816147804260254D-01 BFB4026400000000 -.78161481E-01 /* section 2 A2 */
.1629241704940796D+00 3FC4DAB300000000 .16292417E+00 /* section 3 B0 */
.3258483409881592D+00 3FD4DAB300000000 .32584834E+00 /* section 3 B1 */
.1629241704940796D+00 3FC4DAB300000000 .16292417E+00 /* section 3 B2 */
.5023512840270996D+00 3FE0134300000000 .50235140E+00 /* section 3 A1 */
-.1185797452926636D+00 BFB5B3E000000000 -.11857984E+00 /* section 3 A2 */
.1758027076721191D+00 3FC680B400000000 .17580280E+00 /* section 4 B0 */
.3516055345535278D+00 3FD680B480000000 .35160561E+00 /* section 4 B1 */
.1758027076721191D+00 3FC680B400000000 .17580280E+00 /* section 4 B2 */
.5312651395797729D+00 3FE1001FC0000000 .53126522E+00 /* section 4 A1 */
-.1829618215560913D+00 BFC76B4B00000000 -.18296191E+00 /* section 4 A2 */
.1938179731369019D+00 3FC8CF0700000000 .19381803E+00 /* section 5 B0 */
.3876360654830933D+00 3FD8CF0780000000 .38763607E+00 /* section 5 B1 */
.1938179731369019D+00 3FC8CF0700000000 .19381803E+00 /* section 5 B2 */
.5732599496841431D+00 3FE2582540000000 .57326003E+00 /* section 5 A1 */
-.2764711380004883D+00 BFD1B1B400000000 -.27647125E+00 /* section 5 A2 */
.2187223434448242D+00 3FCBFF1800000000 .21872245E+00 /* section 6 B0 */
.4374448060989380D+00 3FDBFF1800000000 .43744490E+00 /* section 6 B1 */
.2187223434448242D+00 3FCBFF1800000000 .21872245E+00 /* section 6 B2 */
.6320042610168457D+00 3FE4396100000000 .63200432E+00 /* section 6 A1 */
-.4072763919830322D+00 BFDA10D100000000 -.40727646E+00 /* section 6 A2 */
.1969171762466431D+00 3FC9349500000000 .19691723E+00 /* section 7 B0 */
.3938343524932861D+00 3FD9349500000000 .39383447E+00 /* section 7 B1 */
.1969171762466431D+00 3FC9349500000000 .19691723E+00 /* section 7 B2 */
.7132128477096558D+00 3FE6D2A3C0000000 .71321296E+00 /* section 7 A1 */
-.5881026983261108D+00 BFE2D1BCC0000000 -.58810275E+00 /* section 7 A2 */
.3215445280075073D+00 3FD4942F80000000 .32154457E+00 /* section 8 B0 */
.6430890560150146D+00 3FE4942F80000000 .64308913E+00 /* section 8 B1 */
.3215445280075073D+00 3FD4942F80000000 .32154457E+00 /* section 8 B2 */
.8258681297302246D+00 3FEA6D8300000000 .82586822E+00 /* section 8 A1 */
-.8389508724212646D+00 BFEAD8AF80000000 -.83895089E+00 /* section 8 A2 */

```

Note that b1 and b2 generated using the QEDesign 1000 need to be divided by b0, where b0 is included in the total gain.

Plots of the amplitude response, phase response, group delay, impulse response, step response, poles and zeros of the designed low-pass IIR filter are shown in Fig. 7.19.

The IIRLP.ASM assembler program, shown in Fig. 7.20, implements the designed low-pass IIR filter on the DSP56002 processor. Note that this program has “include” statements that reference the STIIR.ASM assembler program shown in Fig. 7.18, the INIT.ASM assembler program shown in Fig. 5.15, and the COEFLP.ASM assembler program shown in Fig. 7.21. A copy of the IIRLP.ASM assembler program, STIIR.ASM assembler program, INIT.ASM assembler program, COEFLP.ASM assembler program, and the IIRLP.CLD absolute object program can be found on the DSP56002 Demonstration Software program diskette.

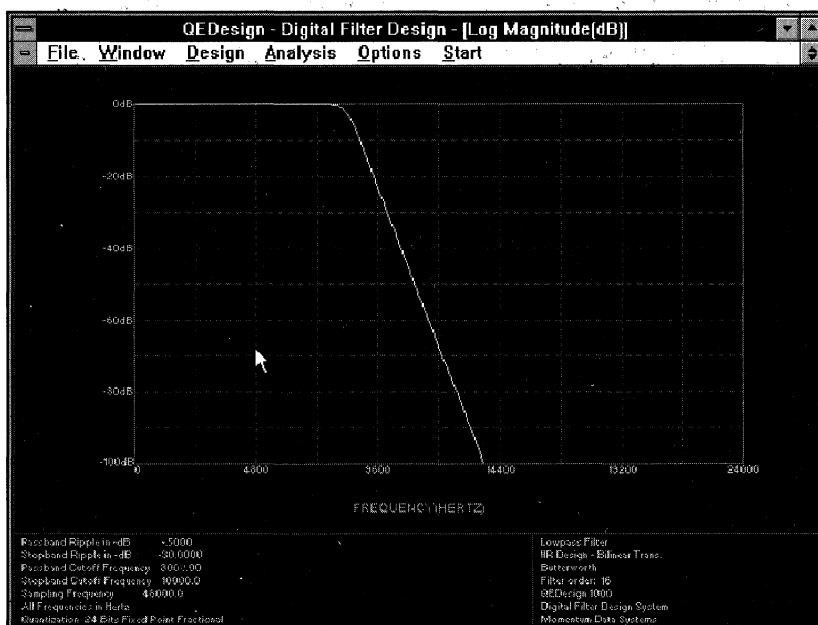


Fig. 7.19 Plots of the Low-pass IIR Filter: a) Amplitude Response

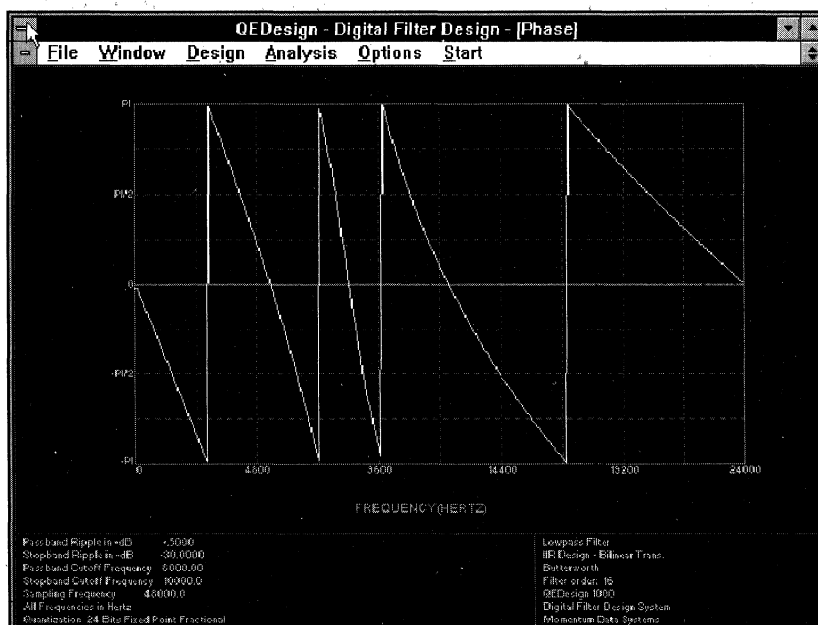


Fig. 7.19 b) Phase Response

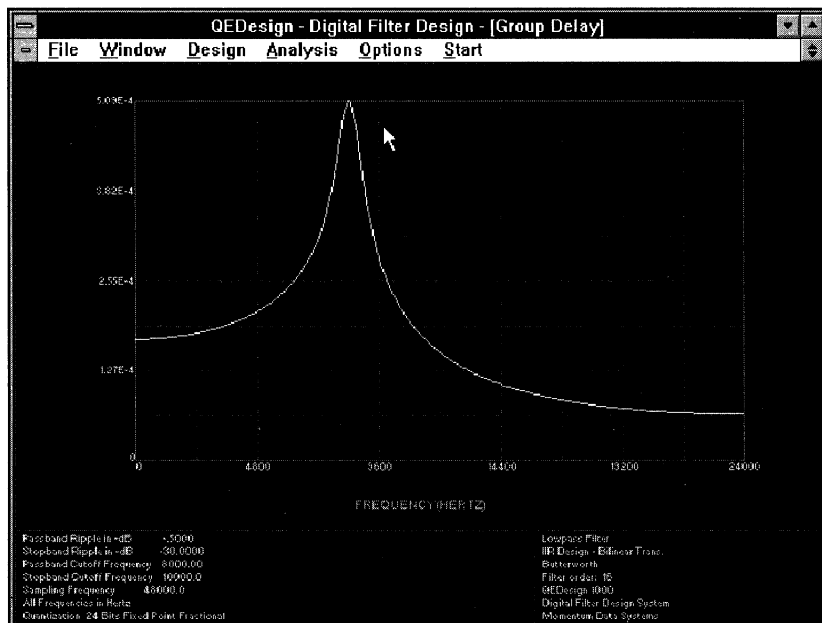


Fig. 7.19 c) Group Delay

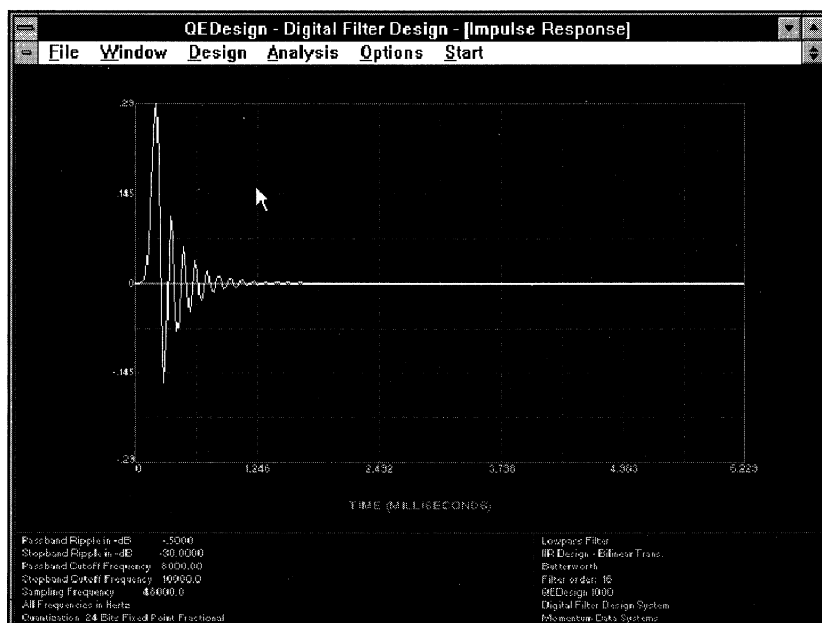


Fig. 7.19 d) Impulse Response

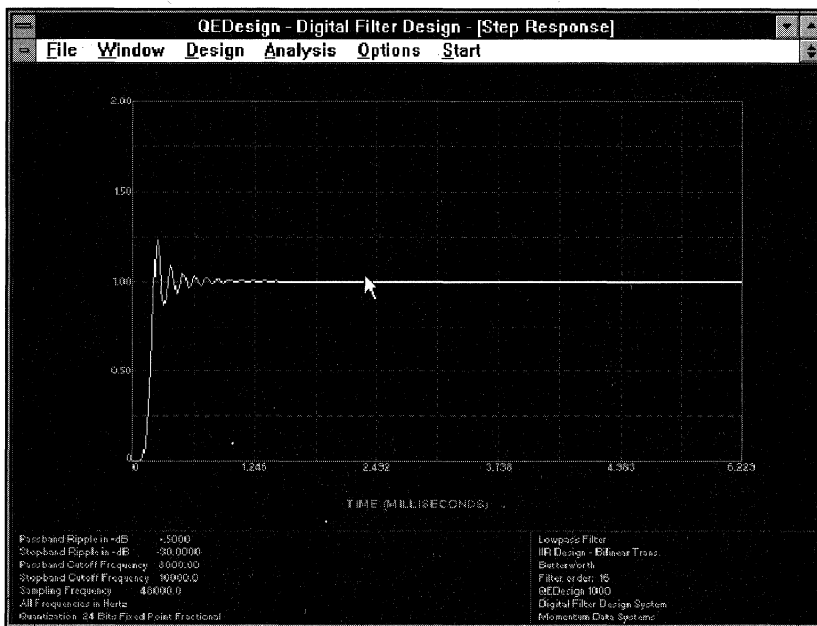


Fig. 7.19 e) Step Response

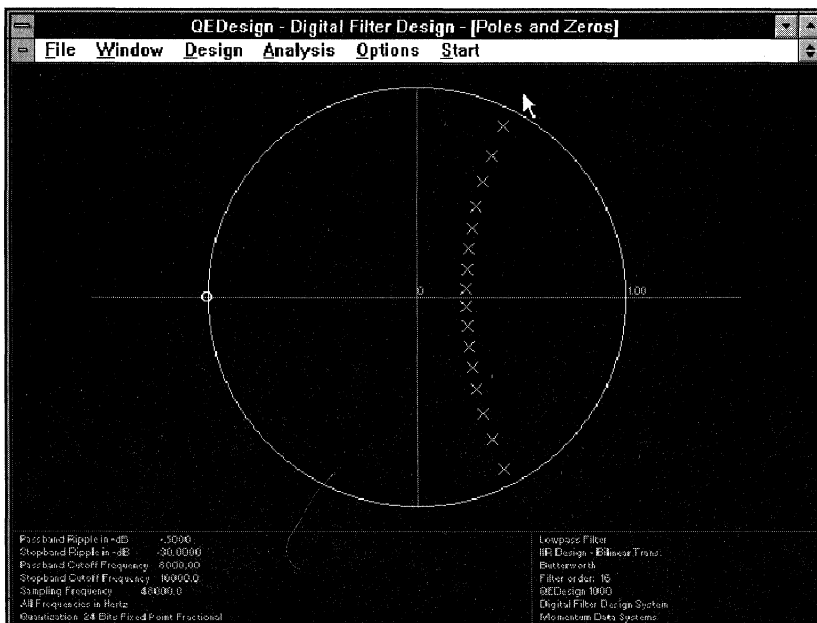


Fig. 7.19 f) Poles and Zeros

Fig. 7.20 IIRLPASM Assembler Program

```

; right channel output = input signal
; left channel output = output of a low pass IIR filter

    include 'init.asm'
    include 'coeflp.asm'
    include 'stiir.asm'

    init_system
    nop

data_loop

    wait_receive
    wait_word

    get_left
    get_right
    nop

    stiir  states,coef,gk,nsec

    put_left
    put_right

    jmp data_loop

```

Fig. 7.21 COEFLPASM Assembler Program

```

;*****
;
; File: COEFLP.ASM
;
;*****
;
;
;*****
; Coefficients of Low Pass IIR
;*****
;
nsec    equ    $8
;
;          org    x:$100
states  ds     2*nsec
;
;          org    x:$120
gk      dc     .10257356E-05
;
;          org    y:$100
coef    dc     -.58660146E-01/2.0
        dc     .47544160/2.0
        dc     1.0000000/2.0
        dc     1.9999999/2.0
;

```

```

dc    -.78161478E-01/2.0
dc    .48419958/2.0
dc    1.0000000/2.0
dc    1.9999999/2.0
;
dc    -.11857983/2.0
dc    .50235134/2.0
dc    1.0000000/2.0
dc    1.9999999/2.0
;
dc    -.18296190/2.0
dc    .53126520/2.0
dc    1.0000000/2.0
dc    1.9999999/2.0
;
dc    -.27647123/2.0
dc    .57326001/2.0
dc    1.0000000/2.0
dc    1.9999999/2.0
;
dc    -.40727645/2.0
dc    .63200432/2.0
dc    1.0000000/2.0
dc    1.9999999/2.0
;
dc    -.58810270/2.0
dc    .71321291/2.0
dc    1.0000000/2.0
dc    1.9999999/2.0
;
dc    -.83895087/2.0
dc    .82586819/2.0
dc    1.0000000/2.0
dc    1.9999999/2.0

```

To demonstrate the designed low-pass IIR filter on the DSP system:

1. Connect a function generator to the left channel of the “in” input on the EVM and to channel one of a dual trace oscilloscope.
2. Set the function generator to produce a 2-volt (peak-to-peak) sine wave.
3. Disconnect the oscilloscope.
4. Connect the two channels of the oscilloscope to “out” left and right channel outputs on the EVM.
5. Load the Debug-EVM Software program.
6. Type change omr 0<return> to change the contents of the OMR register to 0.
7. From within the Debug-EVM program, load the file **iirlp.cld** from the DSP56002 Demonstration Software diskette.
8. Type go<return> to execute the designed low-pass IIR filter program (algorithm) on the DSP56002 processor.
9. Vary the sinusoidal input frequency from 0.5 to 20 KHz.

View the left and right channel analog output signals on the oscilloscope.

Note that the right channel analog output signal is the left channel analog input signal and the left channel analog output signal is the filtered version of the left channel analog input signal. Note that the IIR filter passes the frequencies from 0.5 to 8 KHz and stops the frequencies from 10 to 20 KHz as designed. Also note the increasing attenuation of the left channel analog output signal as the left channel analog input signal's frequency is increased from 8 to 10 KHz.

10. Type `force b<return>` to halt execution of the IIR filter algorithm and return control of the EVM to its monitor program for command entry.
11. Type `quit<return>` to exit the Debug-EVM program.

7.12.2 Designing and Implementing a High-Pass IIR Filter

The following high-pass IIR filter is designed using QEDesign 1000 for Windows:

7.12.2.1 Filter Specifications:

Filter Type:	High-pass
Analog Filter Type:	Chebyshev
Sampling Frequency:	48000 Hz
Passband Cutoff Frequency:	10000 Hz
Stopband Cutoff Frequency:	8000 Hz
Passband Attenuation:	0.5 dB
Stopband Attenuation:	50.0 dB
Filter Design Method:	Bilinear Transformation

7.12.2.2 Design Results

FILTER COEFFICIENT FILE
IIR DESIGN

```

FILTER TYPE                HIGH PASS
ANALOG FILTER TYPE        CHEBYSHEV
PASSBAND RIPPLE IN -dB     -.5000
STOPBAND RIPPLE IN -dB     -50.0000
PASSBAND CUTOFF FREQUENCY  .100000E+05 HERTZ
STOPBAND CUTOFF FREQUENCY  .800000E+04 HERTZ
SAMPLING FREQUENCY        .480000E+05 HERTZ
FILTER DESIGN METHOD:      BILINEAR TRANSFORMATION
FILTER ORDER:             10    000Ah
NUMBER OF SECTIONS:       5     0005h
NO. OF QUANTIZED BITS     24    0018h
QUANTIZATION TYPE:        FRACTIONAL FIXED POINT
COEFFICIENTS SCALED FOR CASCADE FORM II

      -2    FFFFFFFE    /* shift count for overall gain    */
8251735    007DE957    /* overall gain      */
          1    00000001    /* shift count for section 1 values */
915947    000DF9EB    /* section 1 coefficient B0    */
-1831895    FFE40C29    /* section 1 coefficient B1    */
915947    000DF9EB    /* section 1 coefficient B2    */

```

```

-4879875   FFB589FD   /* section 1 coefficient A1   */
-1717038   FFE5CCD2   /* section 1 coefficient A2   */
0          00000000   /* shift count for section 2 values */
3301120    00325F00   /* section 2 coefficient B0   */
-6602240    FF9B4200   /* section 2 coefficient B1   */
3301120    00325F00   /* section 2 coefficient B2   */
-5498668    FFAC18D4   /* section 2 coefficient A1   */
-4566898    FFBA508E   /* section 2 coefficient A2   */
0          00000000   /* shift count for section 3 values */
4139360    003F2960   /* section 3 coefficient B0   */
-8278720    FF81AD40   /* section 3 coefficient B1   */
4139360    003F2960   /* section 3 coefficient B2   */
-727495     FFF4E639   /* section 3 coefficient A1   */
-5917800    FFA5B398   /* section 3 coefficient A2   */
0          00000000   /* shift count for section 4 values */
2082155     001FC56B   /* section 4 coefficient B0   */
-4164310    FFC0752A   /* section 4 coefficient B1   */
2082155     001FC56B   /* section 4 coefficient B2   */
2583059     00276A13   /* section 4 coefficient A1   */
-7033816    FF94AC28   /* section 4 coefficient A2   */
1          00000001   /* shift count for section 5 values */
2789564     002A90BC   /* section 5 coefficient B0   */
-5579128    FFAADE88   /* section 5 coefficient B1   */
2789564     002A90BC   /* section 5 coefficient B2   */
2142240     0020B020   /* section 5 coefficient A1   */
-3975158     FFC3580A   /* section 5 coefficient A2   */

.1091893911361694D+00 3FBBF3D600000000 .21837901E+00 /* section 1 B0 */
-.2183789014816284D+00 BFCBF3D700000000 -.43675801E+00 /* section 1 B1 */
.1091893911361694D+00 3FBBF3D600000000 .21837901E+00 /* section 1 B2 */
-.5817264318466187D+00 BFE29D80C0000000 -.11634530E+01 /* section 1 A1 */
-.2046868801116943D+00 BFCA332E00000000 -.40937392E+00 /* section 1 A2 */
.3935241699218750D+00 3FD92F8000000000 .39352423E+00 /* section 2 B0 */
-.7870483398437500D+00 BFE92F8000000000 -.78704845E+00 /* section 2 B1 */
.3935241699218750D+00 3FD92F8000000000 .39352423E+00 /* section 2 B2 */
-.6554923057556152D+00 BFE4F9CB00000000 -.65549239E+00 /* section 2 A1 */
-.5444166660308838D+00 BFE16BDC80000000 -.54441676E+00 /* section 2 A2 */
.4934501647949219D+00 3FDF94B000000000 .49345018E+00 /* section 3 B0 */
-.9869003295898437D+00 BFEF94B000000000 -.98690035E+00 /* section 3 B1 */
.4934501647949219D+00 3FDF94B000000000 .49345018E+00 /* section 3 B2 */
-.8672416210174561D-01 BFB6338E00000000 -.86724189E-01 /* section 3 A1 */
-.7054567337036133D+00 BFE6931A00000000 -.70545681E+00 /* section 3 A2 */
.2482122182846069D+00 3FCFC56B00000000 .24821227E+00 /* section 4 B0 */
-.4964244365692139D+00 BFDFC56B00000000 -.49642455E+00 /* section 4 B1 */
.2482122182846069D+00 3FCFC56B00000000 .24821227E+00 /* section 4 B2 */
.3079246282577515D+00 3FD3B50980000000 .30792465E+00 /* section 4 A1 */
-.8384962081909180D+00 BFEAD4F600000000 -.83849628E+00 /* section 4 A2 */
.3325419425964355D+00 3FD5485E00000000 .66508391E+00 /* section 5 B0 */
-.6650838851928711D+00 BFE5485E00000000 -.13301678E+01 /* section 5 B1 */
.3325419425964355D+00 3FD5485E00000000 .66508391E+00 /* section 5 B2 */
.2553749084472656D+00 3FD0581000000000 .51074987E+00 /* section 5 A1 */
-.4738757610321045D+00 BFDE53FB00000000 -.94775173E+00 /* section 5 A2 */

```

Plots of the amplitude response; phase response; group delay; impulse response; step response; and poles and zeros of the designed high-pass IIR filter are shown in Fig. 7.22.

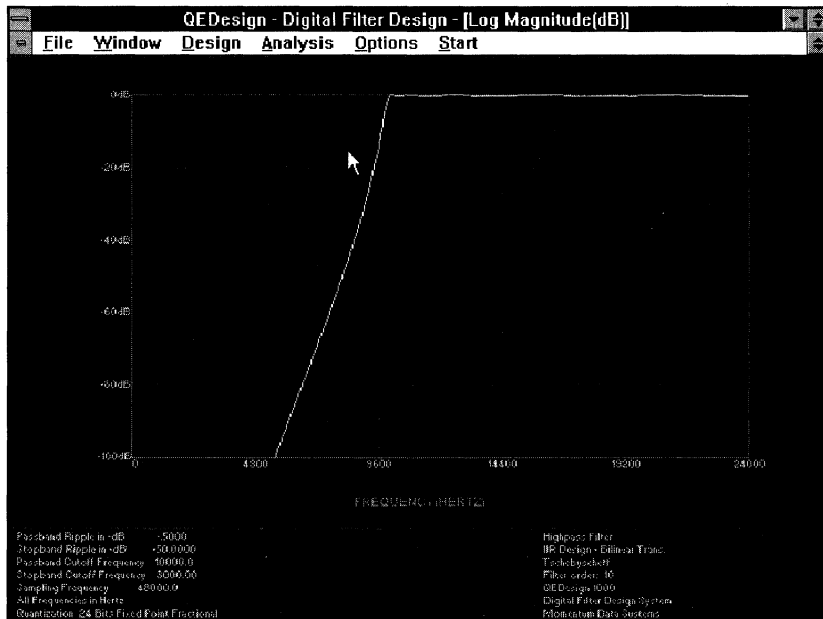


Fig. 7.22 Plots of the High-pass IIR Filter: a) Amplitude Response



Fig. 7.22 b) Phase Response

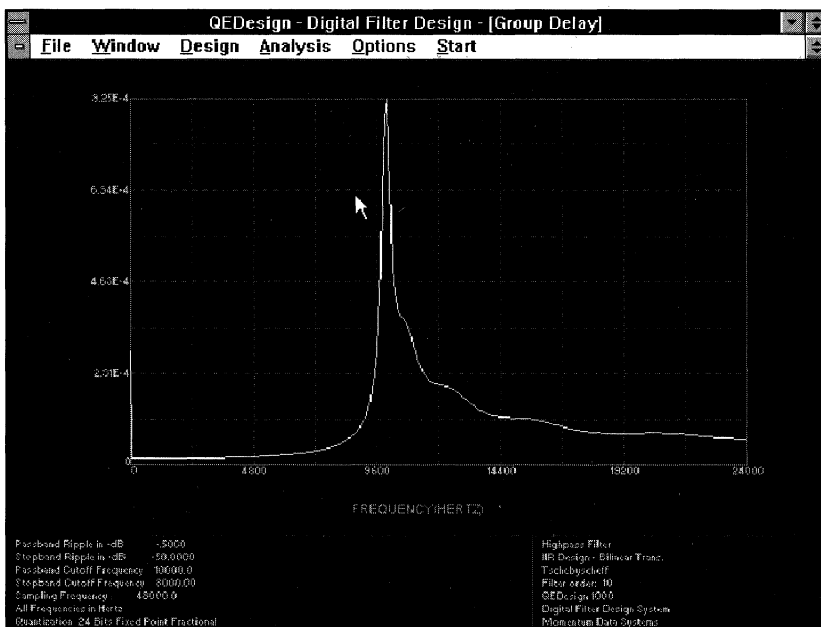


Fig. 7.22 c) Group Delay

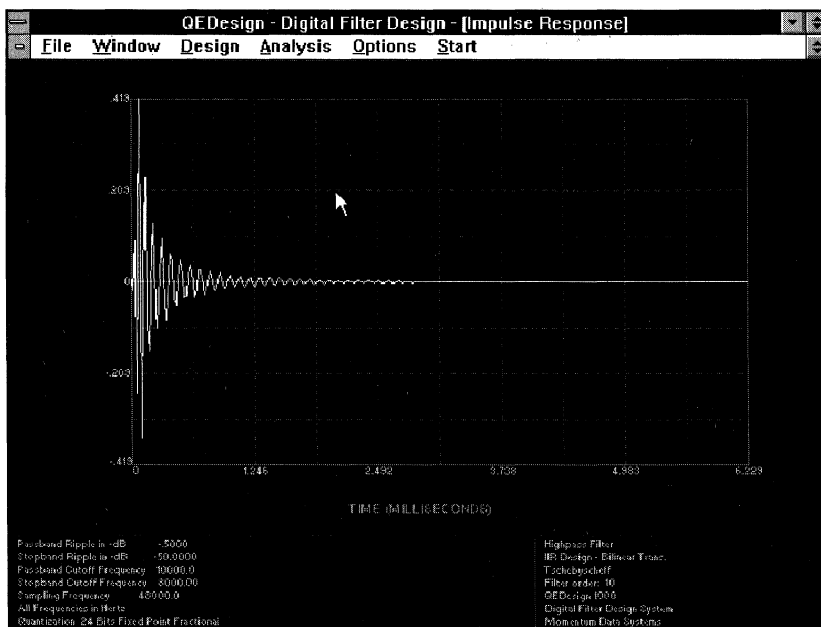


Fig. 7.22 d) Impulse Response

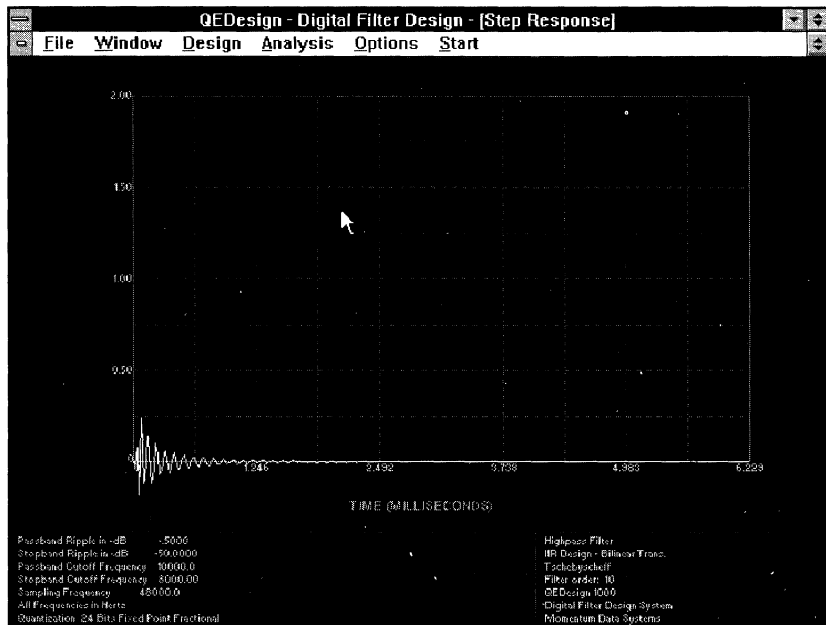


Fig. 7.22 e) Step Response

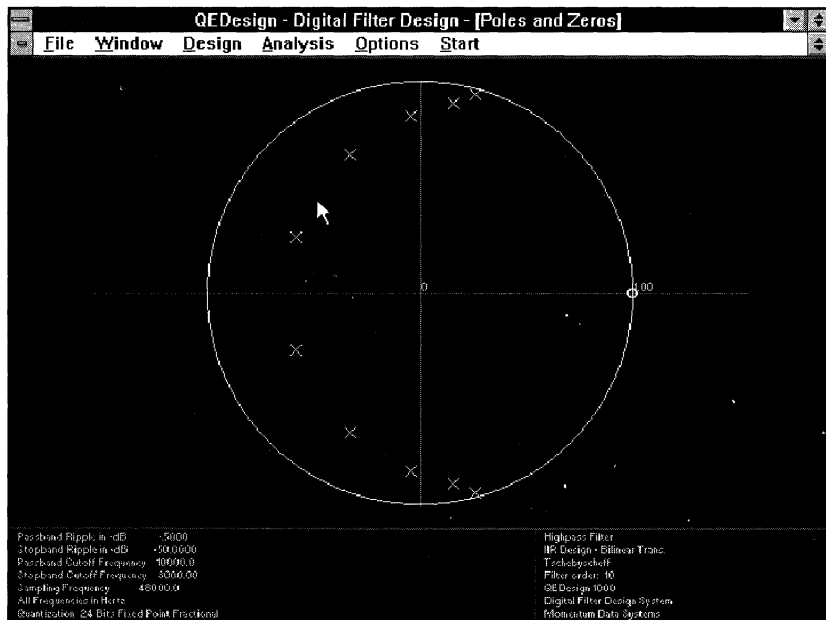


Fig. 7.22 f) Poles and Zeros

The IIRHP.ASM assembler program, shown in Fig. 7.23, implements the designed high-pass IIR filter on the DSP56002 processor. Note that this program has “include” statements that reference the STIIR.ASM assembler program shown in Fig. 7.18, the INIT.ASM assembler program shown in Fig. 5.15, and the COEFHP.ASM assembler program shown in Fig. 7.24. A copy of the IIRHP.ASM assembler program, STIIR.ASM assembler program, INIT.ASM assembler program, COEFHP.ASM assembler program, and the IIRHP.CLD absolute object program can be found on the DSP56002 Demonstration Software program diskette.

Fig. 7.23 IIRHP.ASM Assembler Program

```
; right channel output = input signal
; left channel output = output of a high pass filter

    include 'init.asm'
    include 'coefhp.asm'
    include 'stiir.asm'

    init_system
    nop

data_loop

    wait_receive
    wait_word

    get_left
    get_right
    nop

    stiir states,coef,gk,nsec

    put_left
    put_right

    jmp data_loop
```

Fig. 7.24 COEFHP.ASM Assembler Program

```
;*****
;
; File: COEFHP.ASM
;
;*****
;
;*****
; Coefficients of High Pass IIR filter
;*****
;
nsec    equ    $5
;
;          org    x:$100
```

```

states  ds          2*nsec
;
          org        x:$110
gk      dc          .17215545E-02
;
          org        y:$100
coef
          dc         -.40937391/2.0
          dc         -1.1634530/2.0
          dc         1.0000000/2.0
          dc         -1.9999999/2.0
;
          dc         -.54441673/2.0
          dc         -.65549237/2.0
          dc         1.0000000/2.0
          dc         -1.9999999/2.0
;
          dc         -.70545679/2.0
          dc         -.86724184E-01/2.0
          dc         1.0000000/2.0
          dc         -1.9999999/2.0
;
          dc         -.83849627/2.0
          dc         .30792463/2.0
          dc         1.0000000/2.0
          dc         -1.9999999/2.0
;
          dc         -.94775170/2.0
          dc         .51074982/2.0
          dc         1.0000000/2.0
          dc         -1.9999999/2.0

```

To demonstrate the designed high-pass IIR filter on the DSP System:

1. Connect a function generator to the left channel of the “in” input on the EVM and to channel one of a dual trace oscilloscope.
2. Set the function generator to produce a 2-volt (peak-to-peak) sine wave.
3. Disconnect the oscilloscope.
4. Connect the two channels of the oscilloscope to “out” left and right channel outputs on the EVM.
5. Load the Debug-EVM Software program.
6. Type change omr 0<return> to change the contents of the OMR register to 0.
7. From within the Debug-EVM program, load the file **iirhp.cld** from the DSP56002 Demonstration Software diskette.
8. Type go<return> to execute the designed high-pass IIR filter program (algorithm) on the DSP56002 processor.
9. Vary the sinusoidal input frequency from 0.5 to 20 KHz.

View the left and right analog output signals on the oscilloscope. Note that the right output signal is the input signal and the left analog input signal is the

filtered version of the input signal. Note that the IIR filter stops the frequencies from 0.5 to 8 KHz and passes the frequencies from 10 to 20 KHz as designed. Also note the decreasing attenuation of the output signal as the input signal's frequency is increased from 8 to 10 KHz.

10. Type force b<return> to halt execution of the IIR filter algorithm and return control of the EVM to its monitor program for command entry.
11. Type quit<return> to exit the Debug-EVM program.

7.12.3 Designing and Implementing a Band-Pass IIR Filter

The following band-pass IIR filter is designed using QEDesign 1000 for Windows:

7.12.3.1 Filter Specifications:

Filter Type:	Band-pass
Analog Filter Type:	Butterworth
Sampling Frequency:	48000 Hz
Passband Cutoff Frequencies:	6000 Hz 12000 Hz
Stopband Cutoff Frequencies:	4000 Hz 14000 Hz
Passband Attenuation:	0.5 dB
Stopband Attenuation:	50.0 dB
Filter Design Method:	Bilinear Transformation

7.12.3.2 Design Results

FILTER COEFFICIENT FILE
IIR DESIGN

```

FILTER TYPE                BAND PASS
ANALOG FILTER TYPE        BUTTERWORTH
PASSBAND RIPPLE IN -dB    -.5000
STOPBAND RIPPLE IN -dB    -50.0000
PASSBAND CUTOFF FREQUENCIES .600000E+04 .120000E+05 HERTZ
STOPBAND CUTOFF FREQUENCIES .400000E+04 .140000E+05 HERTZ
SAMPLING FREQUENCY        .480000E+05 HERTZ
FILTER DESIGN METHOD:      BILINEAR TRANSFORMATION
FILTER ORDER              28  001Ch
NUMBER OF SECTIONS        14  000Eh
NO. OF QUANTIZED BITS     24  0018h
QUANTIZATION TYPE:        FRACTIONAL FIXED POINT
COEFFICIENTS SCALED FOR CASCADE FORM II

```

```

      0  00000000  /* shift count for overall gain  */
4788190 00490FDE  /* overall gain  */
      0  00000000  /* shift count for section 1 values  */
2698468 00292CE4  /* section 1 coefficient B0  */
      0  00000000  /* section 1 coefficient B1  */

```

```

-2698468   FFD6D31C   /* section 1 coefficient B2   */
4164302   003F8ACE   /* section 1 coefficient A1   */
-3151874   FFCFE7FE   /* section 1 coefficient A2   */
      0   00000000   /* shift count for section 2 values */
2601857   0027B381   /* section 2 coefficient B0   */
      0   00000000   /* section 2 coefficient B1   */
-2601857   FFD84C7F   /* section 2 coefficient B2   */
2957693   002D217D   /* section 2 coefficient A1   */
-3257039   FFCE4D31   /* section 2 coefficient A2   */
      0   00000000   /* shift count for section 3 values */
2667677   0028B49D   /* section 3 coefficient B0   */
      0   00000000   /* section 3 coefficient B1   */
-2667677   FFD74B63   /* section 3 coefficient B2   */
5451147   00532D8B   /* section 3 coefficient A1   */
-3337316   FFCD139C   /* section 3 coefficient A2   */
      0   00000000   /* shift count for section 4 values */
2675167   0028D1DF   /* section 4 coefficient B0   */
      0   00000000   /* section 4 coefficient B1   */
-2675167   FFD72E21   /* section 4 coefficient B2   */
1904920   001D1118   /* section 4 coefficient A1   */
-3642537   FFC86B57   /* section 4 coefficient A2   */
      0   00000000   /* shift count for section 5 values */
2649737   00286E89   /* section 5 coefficient B0   */
      0   00000000   /* section 5 coefficient B1   */
-2649737   FFD79177   /* section 5 coefficient B2   */
6715848   006679C8   /* section 5 coefficient A1   */
-3770680   FFC676C8   /* section 5 coefficient A2   */
      0   00000000   /* shift count for section 6 values */
2819237   002B04A5   /* section 6 coefficient B0   */
      0   00000000   /* section 6 coefficient B1   */
-2819237   FFD4FB5B   /* section 6 coefficient B2   */
1030614   000FB9D6   /* section 6 coefficient A1   */
-4277312   FFBEBBC0   /* section 6 coefficient A2   */
      0   00000000   /* shift count for section 7 values */
2566913   00272B01   /* section 7 coefficient B0   */
      0   00000000   /* section 7 coefficient B1   */
-2566913   FFD8D4FF   /* section 7 coefficient B2   */
7889494   00786256   /* section 7 coefficient A1   */
-4378393   FFBD30E7   /* section 7 coefficient A2   */
      1   00000001   /* shift count for section 8 values */
1549444   0017A484   /* section 8 coefficient B0   */
      0   00000000   /* section 8 coefficient B1   */
-1549444   FFE85B7C   /* section 8 coefficient B2   */
4478777   00445739   /* section 8 coefficient A1   */
-2551468   FFD91154   /* section 8 coefficient A2   */
      0   00000000   /* shift count for section 9 values */
2700430   0029348E   /* section 9 coefficient B0   */
      0   00000000   /* section 9 coefficient B1   */
-2700430   FFD6CB72   /* section 9 coefficient B2   */
336455    00052247   /* section 9 coefficient A1   */
-5142573   FFB187D3   /* section 9 coefficient A2   */
      1   00000001   /* shift count for section 10 values */
1564638   0017DFDE   /* section 10 coefficient B0  */
      0   00000000   /* section 10 coefficient B1  */

```

```

-1564638   FFE82022   /* section 10 coefficient B2   */
  4967726   004BCD2E   /* section 10 coefficient A1   */
-2959147   FFD2D8D5   /* section 10 coefficient A2   */
    0       00000000   /* shift count for section 11 values */
  2984059   002D887B   /* section 11 coefficient B0   */
    0       00000000   /* section 11 coefficient B1   */
-2984059   FFD27785   /* section 11 coefficient B2   */
  -173267   FFFD5B2D   /* section 11 coefficient A1   */
-6241756   FFA0C224   /* section 11 coefficient A2   */
    1       00000001   /* shift count for section 12 values */
  1067969   00104BC1   /* section 12 coefficient B0   */
    0       00000000   /* section 12 coefficient B1   */
-1067969   FFEFB43F   /* section 12 coefficient B2   */
  5422893   0052BF2D   /* section 12 coefficient A1   */
-3411659   FFCBF135   /* section 12 coefficient A2   */
    0       00000000   /* shift count for section 13 values */
  4087081   003E5D29   /* section 13 coefficient B0   */
    0       00000000   /* section 13 coefficient B1   */
-4087081   FFC1A2D7   /* section 13 coefficient B2   */
  -477310   FFF8B782   /* section 13 coefficient A1   */
-7599728   FF8C0990   /* section 13 coefficient A2   */
    1       00000001   /* shift count for section 14 values */
  2883505   002BFFB1   /* section 14 coefficient B0   */
    0       00000000   /* section 14 coefficient B1   */
-2883505   FFD4004F   /* section 14 coefficient B2   */
  5854240   00595420   /* section 14 coefficient A1   */
-3917151   FFC43AA1   /* section 14 coefficient A2   */

.3216824531555176D+00 3FD4967200000000 .32168250E+00 /* section 1 B0 */
.00000000000000000D+00 0000000000000000 .00000000E+00 /* section 1 B1 */
-.3216824531555176D+00 BFD4967200000000 -.32168250E+00 /* section 1 B2 */
.4964234828948975D+00 3FD4967200000000 .49642355E+00 /* section 1 A1 */
-.3757326602935791D+00 BFD80C0100000000 -.37573270E+00 /* section 1 A2 */
.3101655244827271D+00 3FD3D9C0800000000 .31016553E+00 /* section 2 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 2 B1 */
-.3101655244827271D+00 BFD3D9C0800000000 -.31016553E+00 /* section 2 B2 */
.3525844812393188D+00 3FD690BE800000000 .35258452E+00 /* section 2 A1 */
-.3882693052291870D+00 BFD8D967800000000 -.38826931E+00 /* section 2 A2 */
.3180118799209595D+00 3FD45A4E800000000 .31801190E+00 /* section 3 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 3 B1 */
-.3180118799209595D+00 BFD45A4E800000000 -.31801190E+00 /* section 3 B2 */
.6498273611068726D+00 3FE4CB62C00000000 .64982745E+00 /* section 3 A1 */
-.3978390693664551D+00 BFD97632000000000 -.39783911E+00 /* section 3 A2 */
.3189047574996948D+00 3FD468EF800000000 .31890480E+00 /* section 4 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 4 B1 */
-.3189047574996948D+00 BFD468EF800000000 -.31890480E+00 /* section 4 B2 */
.2270841598510742D+00 3FCD1118000000000 .22708426E+00 /* section 4 A1 */
-.4342242479324341D+00 BFD43744800000000 -.43422428E+00 /* section 4 A2 */
.3158732652664185D+00 3FD43744800000000 .31587327E+00 /* section 5 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 5 B1 */
-.3158732652664185D+00 BFD43744800000000 -.31587327E+00 /* section 5 B2 */
.8005914688110352D+00 3FE99E72000000000 .80059152E+00 /* section 5 A1 */
-.4495000839233398D+00 BFDCC49C000000000 -.44950015E+00 /* section 5 A2 */
.3360792398452759D+00 3FD58252800000000 .33607929E+00 /* section 6 B0 */

```

```

.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 6 B1 */
-.3360792398452759D+00 BFD5825280000000 -.33607929E+00 /* section 6 B2 */
.1228587627410889D+00 3FBF73AC00000000 .12285886E+00 /* section 6 A1 */
-.5098953247070312D+00 BFE0511000000000 -.50989543E+00 /* section 6 A2 */
.3059998750686646D+00 3FD3958080000000 .30599995E+00 /* section 7 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 7 B1 */
-.3059998750686646D+00 BFD3958080000000 -.30599995E+00 /* section 7 B2 */
.9405009746551514D+00 3FEE189580000000 .94050099E+00 /* section 7 A1 */
-.5219451189041138D+00 BFE0B3C640000000 -.52194517E+00 /* section 7 A2 */
.1847081184387207D+00 3FC7A48400000000 .36941624E+00 /* section 8 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 8 B1 */
-.1847081184387207D+00 BFC7A48400000000 -.36941624E+00 /* section 8 B2 */
.5339118242263794D+00 3FE115CE40000000 .10678238E+01 /* section 8 A1 */
-.3041586875915527D+00 BFD3775600000000 -.60831747E+00 /* section 8 A2 */
.3219163417816162D+00 3FD49A4700000000 .32191637E+00 /* section 9 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 9 B1 */
-.3219163417816162D+00 BFD49A4700000000 -.32191637E+00 /* section 9 B2 */
.4010856151580811D-01 3FA4891C00000000 .40108599E-01 /* section 9 A1 */
-.6130424737930298D+00 BFE39E0B40000000 -.61304254E+00 /* section 9 A2 */
.1865193843841553D+00 3FC7DFDE00000000 .37303891E+00 /* section 10 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 10 B1 */
-.1865193843841553D+00 BFC7DFDE00000000 -.37303891E+00 /* section 10 B2 */
.5921990871429443D+00 3FE2F34B80000000 .11843983E+01 /* section 10 A1 */
-.3527578115463257D+00 BFD6939580000000 -.70551580E+00 /* section 10 A2 */
.3557275533676147D+00 3FD6C43D80000000 .35572764E+00 /* section 11 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 11 B1 */
-.3557275533676147D+00 BFD6C43D80000000 -.35572764E+00 /* section 11 B2 */
-.2065503597259521D-01 BF95269800000000 -.20655140E-01 /* section 11 A1 */
-.7440752983093262D+00 BFE7CF7700000000 -.74407536E+00 /* section 11 A2 */
.1273118257522583D+00 3FC04BC100000000 .25462375E+00 /* section 12 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 12 B1 */
-.1273118257522583D+00 BFC04BC100000000 -.25462375E+00 /* section 12 B2 */
.6464592218399048D+00 3FE4AFCB40000000 .12929185E+01 /* section 12 A1 */
-.4067014455795288D+00 BFDA076580000000 -.81340309E+00 /* section 12 A2 */
.4872180223464966D+00 3FDF2E9480000000 .48721807E+00 /* section 13 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 13 B1 */
-.4872180223464966D+00 BFD2E94800000000 -.48721807E+00 /* section 13 B2 */
-.5689978599548340D-01 BFAD21F800000000 -.56899894E-01 /* section 13 A1 */
-.9059581756591797D+00 BFECCFD9C000000000 -.90595821E+00 /* section 13 A2 */
.3437405824661255D+00 3FD5FFD880000000 .68748127E+00 /* section 14 B0 */
.0000000000000000D+00 0000000000000000 .00000000E+00 /* section 14 B1 */
-.3437405824661255D+00 BFD5FFD880000000 -.68748127E+00 /* section 14 B2 */
.6978797912597656D+00 3FE6550800000000 .13957597E+01 /* section 14 A1 */
-.4669607877731323D+00 BFDDE2AF80000000 -.93392168E+00 /* section 14 A2 */

```

Plots of the amplitude response, phase response, group delay, impulse response, step response, poles and zeros of the designed band-pass IIR filter are shown in Fig. 7.25.

The IIRBP.ASM assembler program, shown in Fig. 7.26, implements the designed band-pass IIR filter on the DSP56002 processor. Note that this program has “include” statements that reference the STIIR.ASM assembler program shown in Fig. 7.18, the INIT.ASM assembler program shown in Fig. 5.15, and the COEFBP.ASM assembler program shown in Fig. 7.27. A copy of the IIRBP.ASM assembler program,

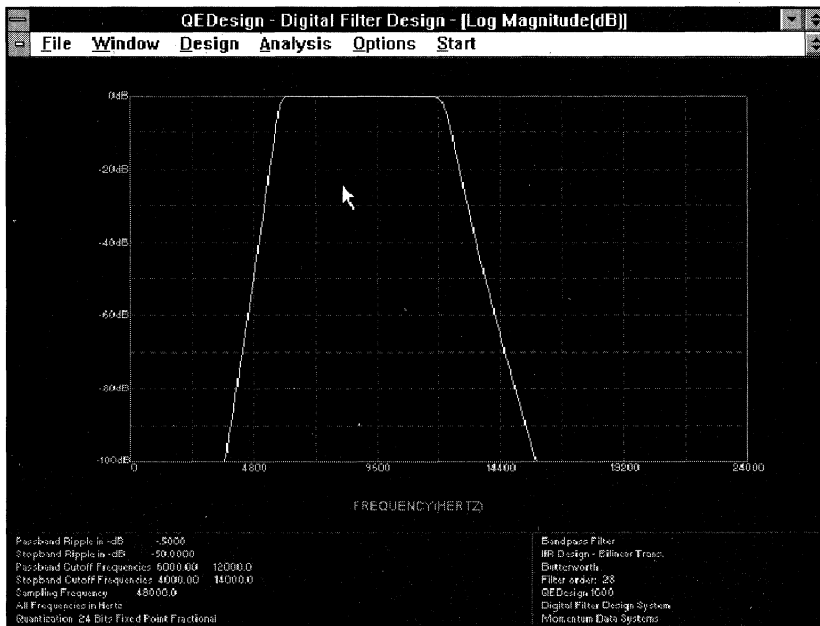


Fig. 7.25 Plots of the Band-pass IIR Filter: a) Amplitude Response

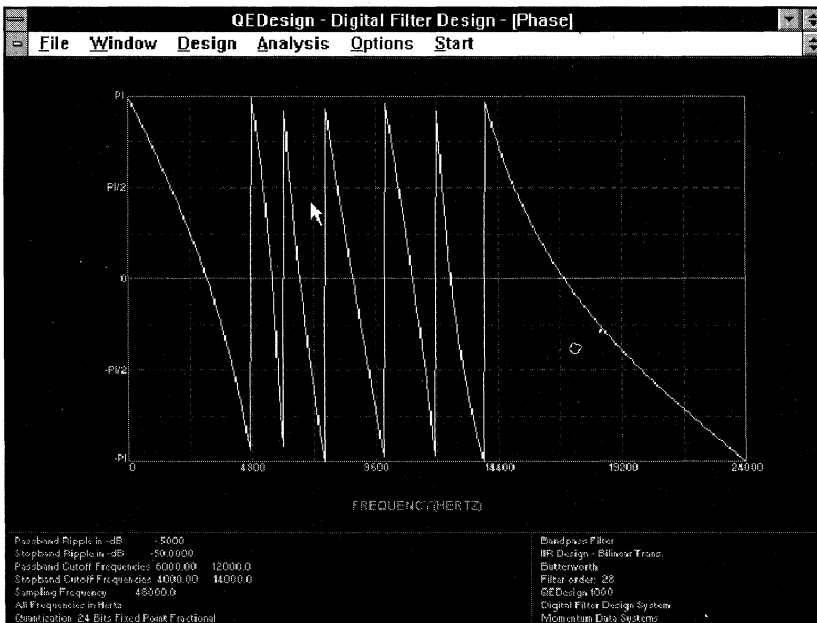


Fig. 7.25 b) Phase Response

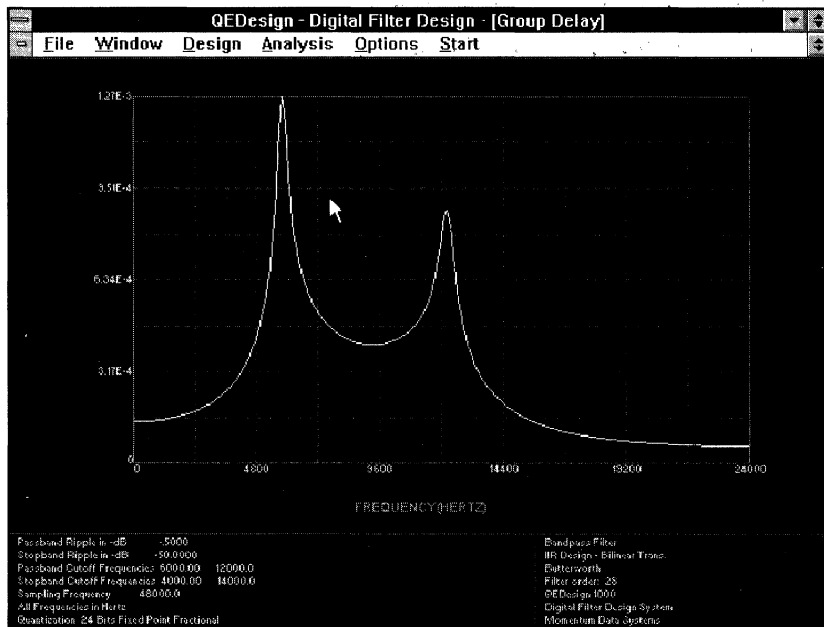


Fig. 7.25 c) Group Delay

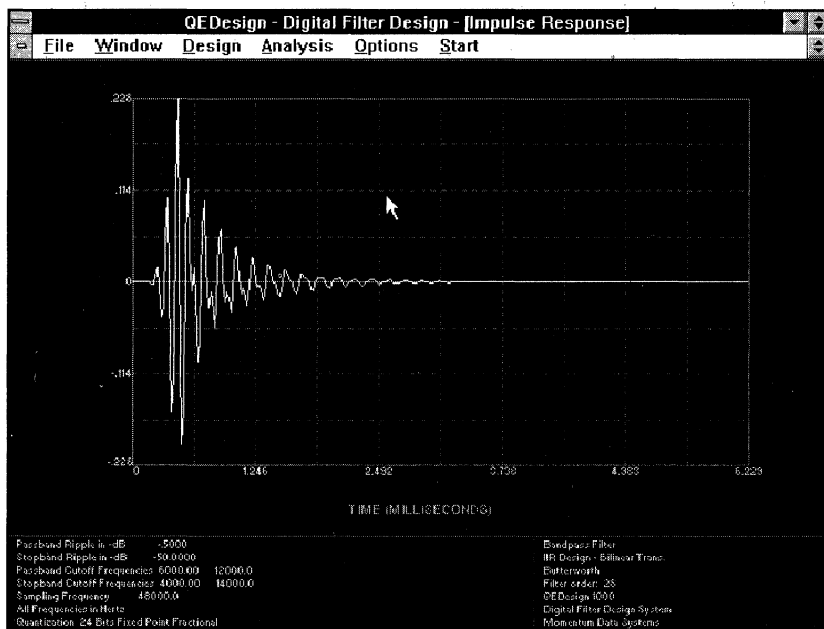


Fig. 7.25 d) Impulse Response

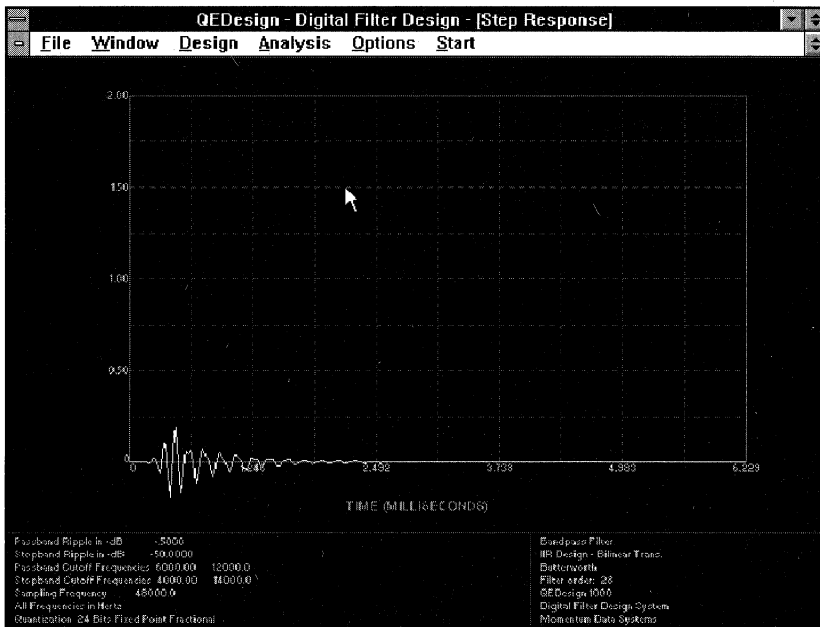


Fig. 7.25 e) Step Response

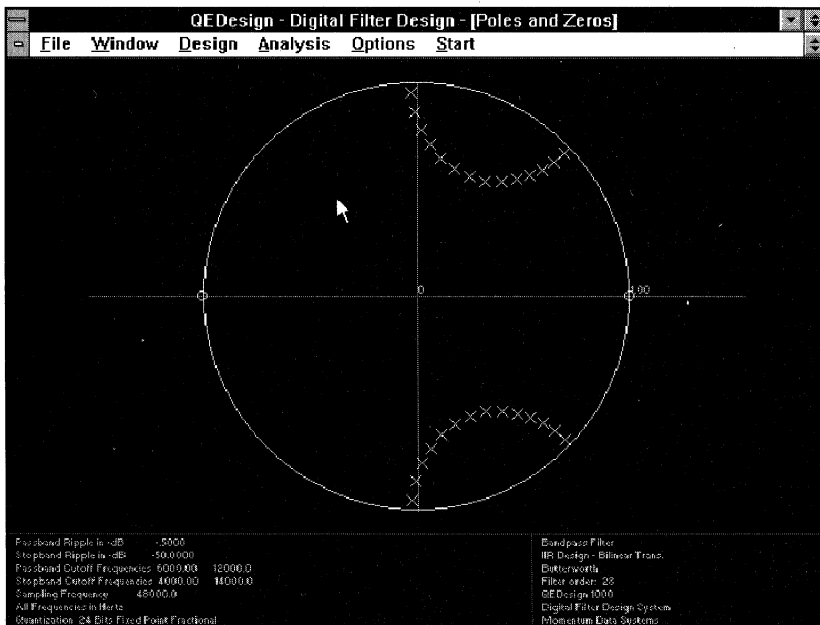


Fig. 7.25 f) Poles and Zeros

STIIR.ASM assembler program, INIT.ASM assembler program, COEFBP.ASM assembler program, and the IIRBP.CLD absolute object program can be found on the DSP56002 Demonstration Software program diskette.

Fig. 7.26 IIRBP.ASM Assembler Program

```
; right channel output = input signal
; left channel output = output of a band pass filter

    include 'init.asm'
    include 'coefbp.asm'
    include 'stiir.asm'

    init_system
    nop

data_loop

    wait_receive
    wait_word

    get_left
    get_right
    nop

    stiir states,coef,gk,nsec

    put_left
    put_right

    jmp data_loop
```

Fig. 7.27 COEFBP.ASM Assembler Program

```
;*****
;
; File: COEFBP.ASM
;
;*****
;
;
;*****
; Coefficients of Band Pass IIR filter
;*****
;
nsec    equ        $e
;
;        org        x:$100
states  ds          2*nsec
;
;        org        x:$120
gk      dc          .25252064E-06
;
;        org        y:$100
coef
```

```
dc -.37573269/2.0
dc .49642354/2.0
dc -1.0000000/2.0
dc .00000000/2.0
;
dc -.38826931/2.0
dc .35258451/2.0
dc -1.0000000/2.0
dc .00000000/2.0
;
dc -.39783910/2.0
dc .64982742/2.0
dc -1.0000000/2.0
dc .00000000/2.0
;
dc -.43422428/2.0
dc .22708425/2.0
dc -1.0000000/2.0
dc .00000000/2.0
;
dc -.44950014/2.0
dc .80059147/2.0
dc -1.0000000/2.0
dc .00000000/2.0
;
dc -.50989538/2.0
dc .12285886/2.0
dc -1.0000000/2.0
dc .00000000/2.0
;
dc -.52194512/2.0
dc .94050097/2.0
dc -1.0000000/2.0
dc .00000000/2.0
;
dc -.60831743/2.0
dc 1.0678238/2.0
dc -1.0000000/2.0
dc .00000000/2.0
;
dc -.61304253/2.0
dc .40108599E-1/2.0
dc -1.0000000/2.0
dc .00000000/2.0
;
dc -.70551580/2.0
dc 1.1843983/2.0
dc -1.0000000/2.0
dc .00000000/2.0
;
dc -.74407536/2.0
dc -.20655140E-1/2.0
dc -1.0000000/2.0
dc .00000000/2.0
```

```

;
    dc    -.81340307/2.0
    dc    1.2929184/2.0
    dc    -1.0000000/2.0
    dc    .00000000/2.0
;
    dc    -.90595818/2.0
    dc    -.56899894E-1/2.0
    dc    -1.0000000/2.0
    dc    .00000000/2.0
;
    dc    -.93392164/2.0
    dc    1.3957597/2.0
    dc    -1.0000000/2.0
    dc    .00000000/2.0

```

To demonstrate the designed band-stop IIR filter on the DSP System:

1. Connect a function generator to the left channel of the “in” input on the EVM and to channel one of a dual trace oscilloscope.
2. Set the function generator to produce a 2-volt (peak-to-peak) sine wave.
3. Disconnect the oscilloscope.
4. Connect the two channels of the oscilloscope to “out” left and right channel outputs on the EVM.
5. Load the Debug-EVM software program.
6. Type `change omr 0<return>` to change the contents of the OMR register to 0.
7. From within the Debug-EVM program, load the file **iirbp.cld** from the DSP56002 Demonstration Software diskette.
8. Type `go<return>` to execute the designed band-pass IIR filter program (algorithm) on the DSP56002 processor.
9. Vary the sinusoidal input frequency from 0.5 to 20 KHz.

View the left and right analog output signals on the oscilloscope. Note that the right output signal is the input signal and the left analog input signal is the filtered version of the input signal. Note that the IIR filter passes the frequencies from 6 to 12 KHz and stops the frequencies from 0.5 to 4 KHz and 14 to 20 KHz as designed. Also note the decreasing attenuation of the output signal as the input signal's frequency is increased from 4 to 6 KHz; note the increasing attenuation of the output signal as the input signal's frequency is increased from 12 to 14 KHz.

10. Type `force b<return>` to halt execution of the IIR filter algorithm and return control of the EVM to its monitor program for command entry.
11. Type `quit<return>` to exit the Debug-EVM program.

7.12.4 Designing and Implementing a Band-Stop IIR Filter

The following band-pass IIR filter is designed using QEDesign 1000 for Windows:

7.12.4.1 Filter Specifications:

Filter Type:	Band-stop
Analog Filter Type:	Chebyshev
Sampling Frequency:	48000 Hz
Passband Cutoff Frequencies:	4000 Hz 14000 Hz
Stopband Cutoff Frequencies:	6000 Hz 12000 Hz
Passband Attenuation:	0.5 dB
Stopband Attenuation:	50.0 dB
Filter Design Method:	Bilinear Transformation

7.12.4.2 Design Results

FILTER COEFFICIENT FILE
IIR DESIGN

```

FILTER TYPE                BAND STOP
ANALOG FILTER TYPE        CHEBYSHEV
PASSBAND RIPPLE IN -dB    -.5000
STOPBAND RIPPLE IN -dB    -50.0000
PASSBAND CUTOFF FREQUENCIES .400000E+04 .140000E+05 HERTZ
STOPBAND CUTOFF FREQUENCIES .600000E+04 .120000E+05 HERTZ
SAMPLING FREQUENCY        .480000E+05 HERTZ
FILTER DESIGN METHOD:      BILINEAR TRANSFORMATION
FILTER ORDER              16  0010h
NUMBER OF SECTIONS        8   0008h
NO. OF QUANTIZED BITS     24  0018h
QUANTIZATION TYPE:        FRACTIONAL FIXED POINT
COEFFICIENTS SCALED FOR CASCADE FORM II

```

```

-2  FFFFFFFF /* shift count for overall gain */
7386506 0070B58A /* overall gain */
1  00000001 /* shift count for section 1 values */
981822 000EFB3E /* section 1 coefficient B0 */
-947187 FFF18C0D /* section 1 coefficient B1 */
981822 000EFB3E /* section 1 coefficient B2 */
-5147614 FFB17422 /* section 1 coefficient A1 */
-1877662 FFE35962 /* section 1 coefficient A2 */
1  00000001 /* shift count for section 2 values */
4945854 004B77BE /* section 2 coefficient B0 */
-4771383 FFB731C9 /* section 2 coefficient B1 */
4945854 004B77BE /* section 2 coefficient B2 */
-3877197 FFC4D6B3 /* section 2 coefficient A1 */
-2713354 FFD698F6 /* section 2 coefficient A2 */
1  00000001 /* shift count for section 3 values */
1484495 0016A6CF /* section 3 coefficient B0 */
-1432127 FFEA25C1 /* section 3 coefficient B1 */
1484495 0016A6CF /* section 3 coefficient B2 */
7194555 006DC7BB /* section 3 coefficient A1 */
-3146274 FFCFFDDE /* section 3 coefficient A2 */

```

```

1      00000001      /* shift count for section 4 values */
5398638 0052606E      /* section 4 coefficient B0 */
-5208195 FFB0877D      /* section 4 coefficient B1 */
5398638 0052606E      /* section 4 coefficient B2 */
-2666602 FFD74F96      /* section 4 coefficient A1 */
-3454725 FFCB48FB      /* section 4 coefficient A2 */
1      00000001      /* shift count for section 5 values */
1859675 001C605B      /* section 5 coefficient B0 */
-1794072 FFE49FE8      /* section 5 coefficient B1 */
1859675 001C605B      /* section 5 coefficient B2 */
7168133 006D6085      /* section 5 coefficient A1 */
-3482475 FFCADC95      /* section 5 coefficient A2 */
1      00000001      /* shift count for section 6 values */
2438221 0025344D      /* section 6 coefficient B0 */
-2352209 FFD C1BAF      /* section 6 coefficient B1 */
2438221 0025344D      /* section 6 coefficient B2 */
7118825 006C9FE9      /* section 6 coefficient A1 */
-3815546 FFC5C786      /* section 6 coefficient A2 */
0      00000000      /* shift count for section 7 values */
6420556 0061F84C      /* section 7 coefficient B0 */
-6194063 FFA17C71      /* section 7 coefficient B1 */
6420556 0061F84C      /* section 7 coefficient B2 */
-4161017 FFC08207      /* section 7 coefficient A1 */
-7938910 FF86DCA2      /* section 7 coefficient A2 */
1      00000001      /* shift count for section 8 values */
5422032 0052BBD0      /* section 8 coefficient B0 */
-5230764 FFB02F54      /* section 8 coefficient B1 */
5422032 0052BBD0      /* section 8 coefficient B2 */
7153544 006D2788      /* section 8 coefficient A1 */
-4076088 FFC1CDC8      /* section 8 coefficient A2 */

.1170423030853271D+00 3FBDF67C00000000 .23408464E+00 /* section 1 B0 */
- .1129134893417358D+00 BFBCE7E600000000 - .22582702E+00 /* section 1 B1 */
.1170423030853271D+00 3FBDF67C00000000 .23408464E+00 /* section 1 B2 */
- .6136434078216553D+00 BFE3A2F780000000 - .12272870E+01 /* section 1 A1 */
- .2238347530364990D+00 BFCCA69E00000000 - .44766951E+00 /* section 1 A2 */
.5895917415618896D+00 3FE2DDEF80000000 .11791836E+01 /* section 2 B0 */
- .5687931776046753D+00 BFE2338DC0000000 - .11375865E+01 /* section 2 B1 */
.5895917415618896D+00 3FE2DDEF80000000 .11791836E+01 /* section 2 B2 */
- .4621978998184204D+00 BFDD94A680000000 - .92439592E+00 /* section 2 A1 */
- .3234570026397705D+00 BFD4B38500000000 - .64691420E+00 /* section 2 A2 */
.1769655942916870D+00 3FC6A6CF00000000 .35393121E+00 /* section 3 B0 */
- .1707228422164917D+00 BFC5DA3F00000000 - .34144587E+00 /* section 3 B1 */
.1769655942916870D+00 3FC6A6CF00000000 .35393121E+00 /* section 3 B2 */
.8576577901840210D+00 3FEB71EEC0000000 .17153156E+01 /* section 3 A1 */
- .3750650882720947D+00 BFD8011100000000 - .75013041E+00 /* section 3 A2 */
.6435678005218506D+00 3FE4981B80000000 .12871357E+01 /* section 4 B0 */
- .6208652257919312D+00 BFE3DE20C0000000 - .12417305E+01 /* section 4 B1 */
.6435678005218506D+00 3FE4981B80000000 .12871357E+01 /* section 4 B2 */
- .3178837299346924D+00 BFD4583500000000 - .63576761E+00 /* section 4 A1 */
- .4118353128433228D+00 BFDA5B8280000000 - .82367080E+00 /* section 4 A2 */
.2216905355453491D+00 3FCC605B00000000 .44338110E+00 /* section 5 B0 */
- .2138700485229492D+00 BFCEB60180000000 - .42774031E+00 /* section 5 B1 */
.2216905355453491D+00 3FCC605B00000000 .44338110E+00 /* section 5 B2 */

```



```

.8545080423355103D+00 3FEB582140000000 .17090161E+01 /* section 5 A1 */
-.4151433706283569D+00 BFDA91B580000000 -.83028684E+00 /* section 5 A2 */
.2906585931777954D+00 3FD29A2680000000 .58131721E+00 /* section 6 B0 */
-.2804051637649536D+00 BFD1F22880000000 -.56081056E+00 /* section 6 B1 */
.2906585931777954D+00 3FD29A2680000000 .58131721E+00 /* section 6 B2 */
.8486300706863403D+00 3FEB27FA40000000 .16972602E+01 /* section 6 A1 */
-.4548485279083252D+00 BFDD1C3D00000000 -.90969719E+00 /* section 6 A2 */
.7653899192810059D+00 3FE87E1300000000 .76538998E+00 /* section 7 B0 */
-.7383898496627808D+00 BFE7A0E3C0000000 -.73838994E+00 /* section 7 B1 */
.7653899192810059D+00 3FE87E1300000000 .76538998E+00 /* section 7 B2 */
-.4960318803787231D+00 BFDFBEEFC800000000 -.49603196E+00 /* section 7 A1 */
-.9463918209075928D+00 BFEE48D78000000000 -.94639194E+00 /* section 7 A2 */
.6463565826416016D+00 3FE4AEF400000000 .12927133E+01 /* section 8 B0 */
-.6235556602478027D+00 BFE3F42B0000000000 -.12471113E+01 /* section 8 B1 */
.6463565826416016D+00 3FE4AEF400000000 .12927133E+01 /* section 8 B2 */
.8527688980102539D+00 3FEB49E200000000 .17055379E+01 /* section 8 A1 */
-.4859075546264648D+00 BFDF191C0000000000 -.97181511E+00 /* section 8 A2 */

```

Plots of the amplitude response; phase response; group delay; impulse response; step response; and poles and zeros of the designed band-stop IIR filter are shown in Fig. 7.28.

The IIRBS.ASM assembler program, shown in Fig. 7.29, implements the designed band-stop IIR filter on the DSP56002 processor. Note that this program has “include” statements that reference the STIIR.ASM assembler program shown in Fig.

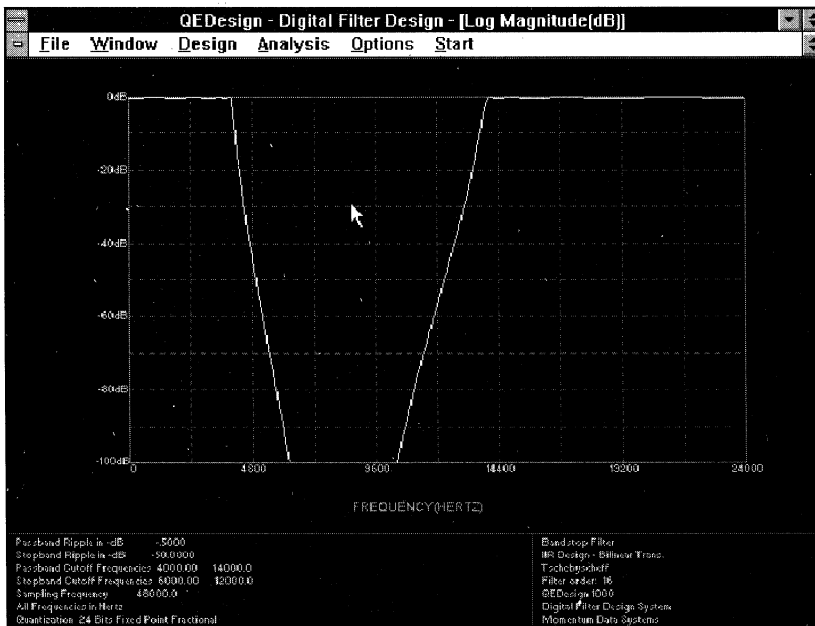


Fig. 7.28 Plots of the Band-stop IIR Filter: a) Amplitude Response

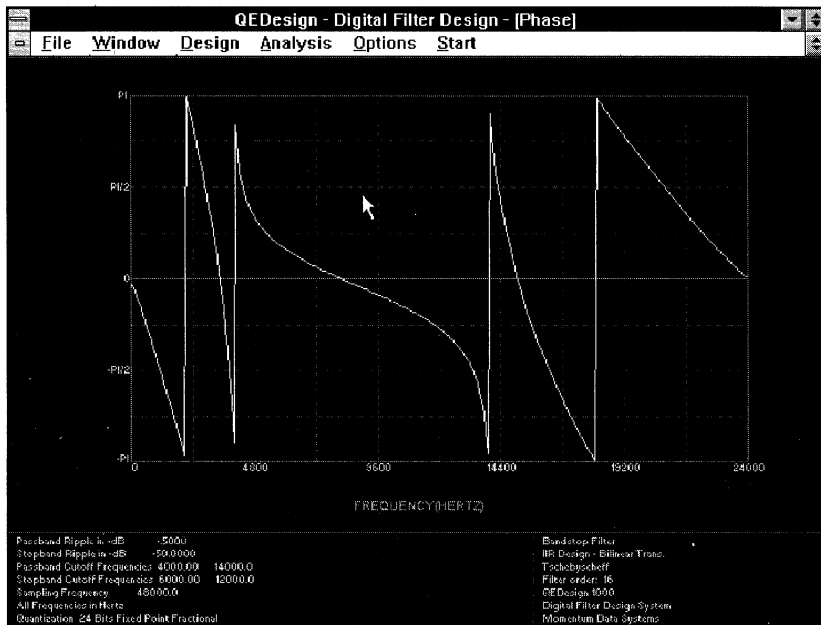


Fig. 7.28 b) Phase Response

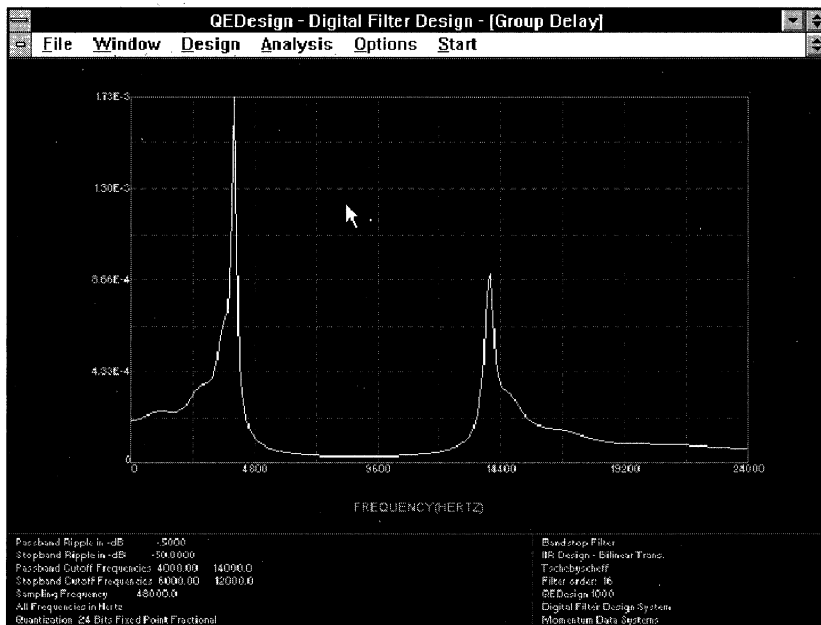


Fig. 7.28 c) Group Delay

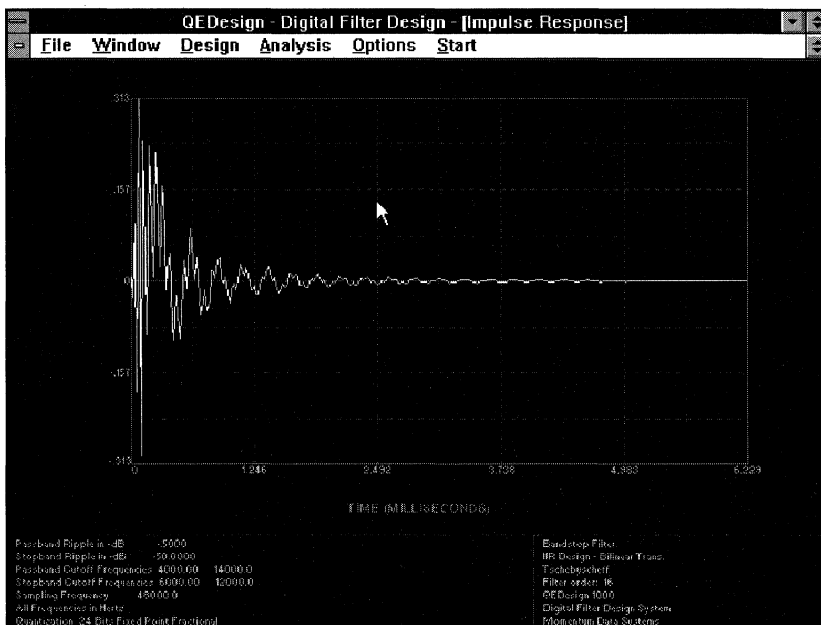


Fig. 7.28 d) Impulse Response

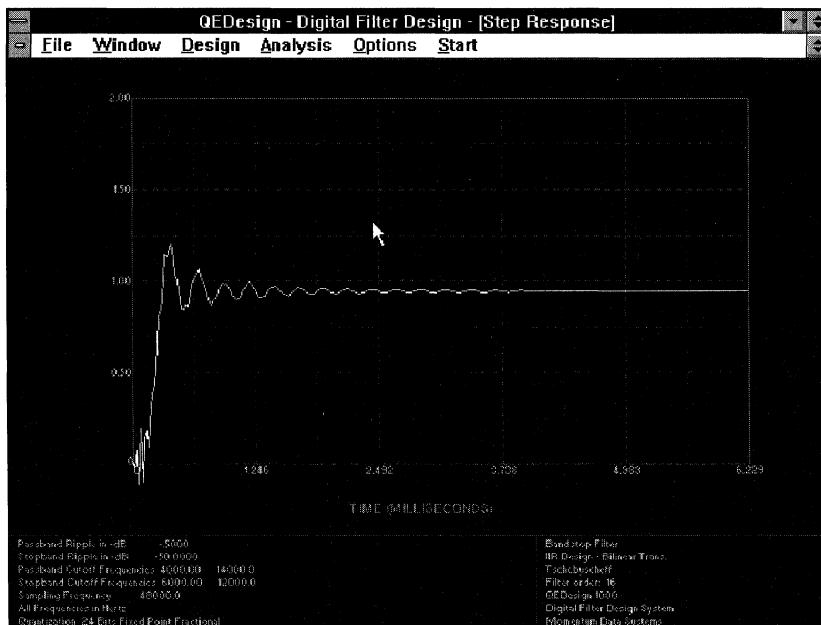


Fig. 7.28 e) Step Response

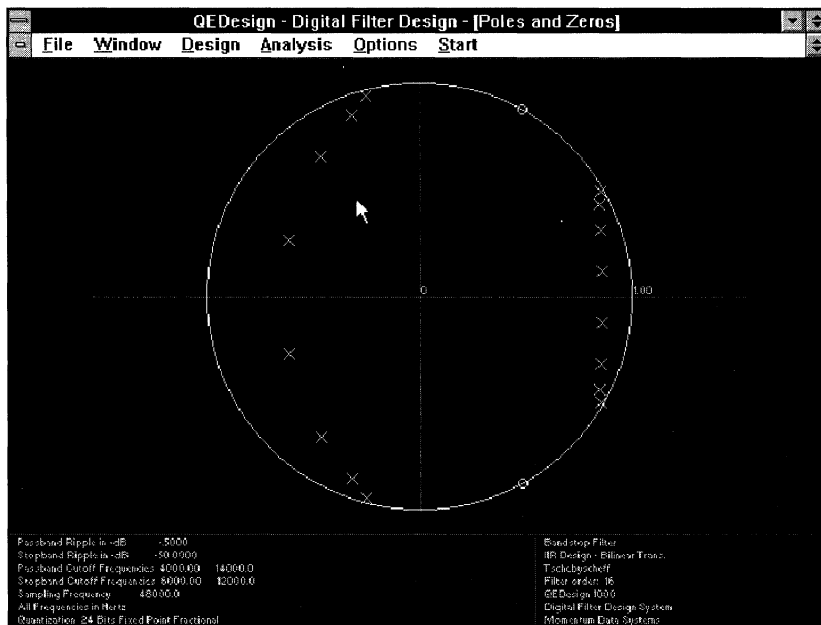


Fig. 7.28 f) Poles and Zeros

7.18, the INIT.ASM assembler program shown in Fig. 5.15, and the COEFBS.ASM assembler program shown in Fig. 7.30. A copy of the IIRBS.ASM assembler program, STIIR.ASM assembler program, INIT.ASM assembler program, COEFBS.ASM assembler program, and the IIRBS.CLD absolute object program can be found on the DSP56002 Demonstration Software program diskette.

Fig. 7.29 IIRBS.ASM Assembler Program

```
; right channel output = input signal
; left channel output = output of a band-stop filter

include 'init.asm'
include 'coefbs.asm'
include 'stiir.asm'

init_system
nop

data_loop

wait_receive
wait_word

get_left
get_right
nop

stiir states,coef,gk,nsec
```

```

put_left
put_right

jmp data_loop

```

Fig. 7.30 COEFBS.ASM Assembler Program

```

;*****
;
; File: COEFBS.ASM
;
;*****
;
;*****
; Coefficients of Band Stop IIR filter
;*****
;
nsec    equ        $8
;
;          org      x:$100
states  ds         2*nsec
;
;          org      x:$120
gk       dc        .70593073E-02
;
;          org      y:$100
coef
;          dc       -.44766951/2.0
;          dc       -1.2272869/2.0
;          dc       1.0000000/2.0
;          dc       -.96472377/2.0
;
;          dc       -.64691418/2.0
;          dc       -.92439586/2.0
;          dc       1.0000000/2.0
;          dc       -.96472377/2.0
;
;          dc       -.75013036/2.0
;          dc       1.7153156/2.0
;          dc       1.0000000/2.0
;          dc       -.96472377/2.0
;
;          dc       -.82367074/2.0
;          dc       -.63576758/2.0
;          dc       1.0000000/2.0
;          dc       -.96472377/2.0
;
;          dc       -.83028680/2.0
;          dc       1.7090161/2.0
;          dc       1.0000000/2.0
;          dc       -.96472377/2.0
;
;          dc       -.90969718/2.0

```

```

dc      1.6972601/2.0
dc      1.0000000/2.0
dc      -.96472377/2.0
;
dc      -.94639188/2.0
dc      -.49603194/2.0
dc      1.0000000/2.0
dc      -.96472377/2.0
;
dc      -.97181511/2.0
dc      1.7055378/2.0
dc      1.0000000/2.0
dc      -.96472377/2.0

```

To demonstrate the designed band-stop IIR filter on the DSP System:

1. Connect a function generator to the left channel of the “in” input on the EVM and to channel one of a dual trace oscilloscope.
2. Set the function generator to produce a 2-volt (peak-to-peak) sine wave.
3. Disconnect the oscilloscope.
4. Connect the two channels of the oscilloscope to “out” left and right channel outputs on the EVM.
5. Load the Debug-EVM Software program.
6. Type `change omr 0<return>` to change the contents of the OMR register to 0.
7. From within the Debug-EVM program, load the file **iirbs.cld** from the DSP56002 Demonstration Software diskette.
8. Type `go<return>` to execute the designed band-stop IIR filter program (algorithm) on the DSP56002 processor.
9. Vary the sinusoidal input frequency from 0.5 to 20 KHz.

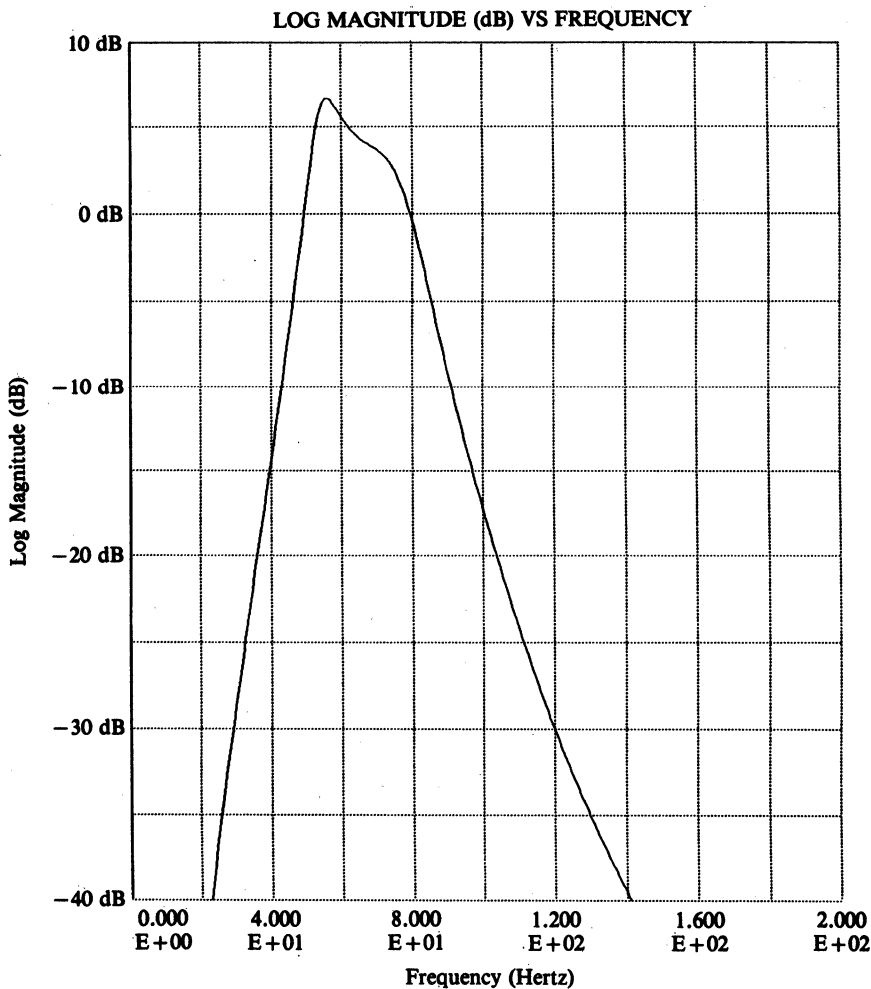
View the left and right analog output signals on the oscilloscope. Note that the right output signal is the input signal and the left analog input signal is the filtered version of the input signal. Note that the IIR filter stops the frequencies from 6 to 12 KHz and passes the frequencies from 0.5 to 4 KHz and 14 to 20 KHz as designed. Also note the increasing attenuation of the output signal as the input signal's frequency is increased from 4 to 6 KHz; note the decreasing attenuation of the output signal as the input signal's frequency is increased from 12 to 14 KHz.

10. Type `force b<return>` to halt execution of the IIR filter algorithm and return control of the EVM to its monitor program for command entry.
11. Type `quit<return>` to exit the Debug-EVM program.

7.13 QUANTIZATION EFFECTS

The Digital Filter Design and Analysis System Software program can also be used to compare the effects of quantization, e.g., the word length precision of the digital signal

processor. Figure 7.31 shows the frequency response of a tenth order 50 Hz bandpass IIR filter implemented with a 24-bit DSP56002 processor and a 16-bit processor. It is clear that the filter implemented with the 24-bit DSP56002 processor has a better performance than the filter implemented with a 16-bit processor.

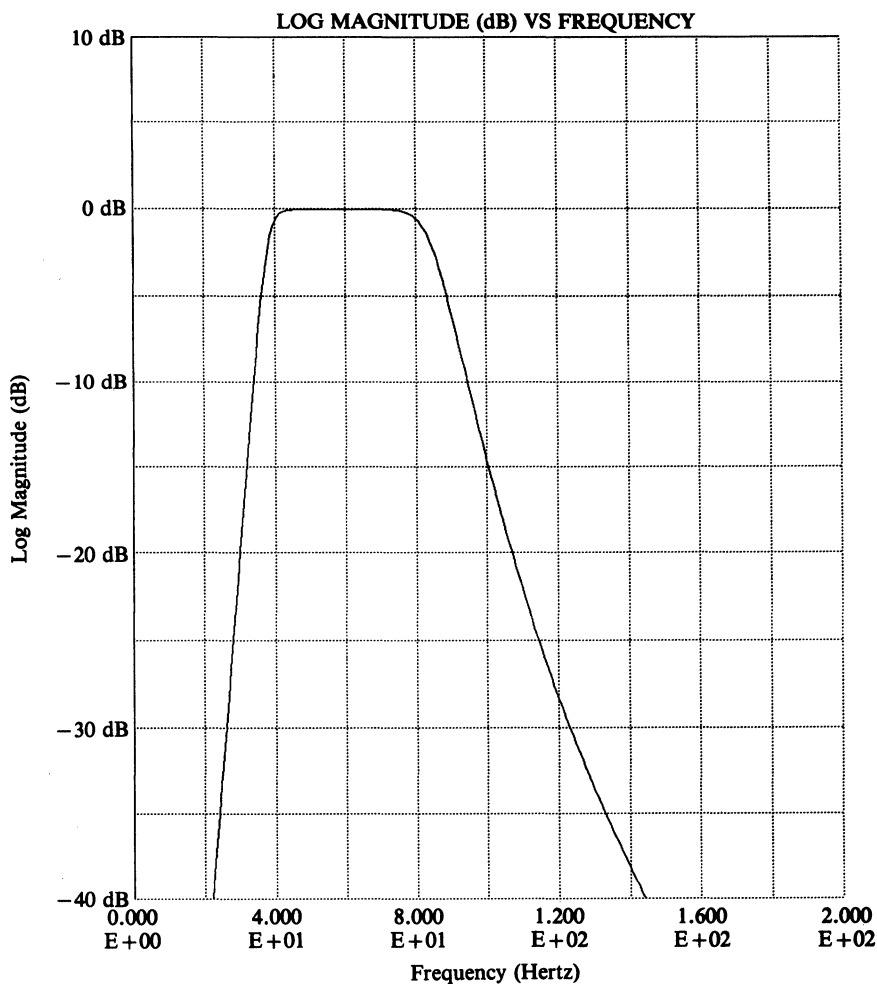


Passband Ripple in -dB	-1.0000
Stopband Ripple in -dB	-30.0000
Passband Cutoff Frequencies	40.0000 82.5000
Stopband Cutoff Frequencies	20.0000 125.000
Sampling Frequency	44100.0
All Frequencies in Hertz	
Quantization 16 Bits Fixed Point Fractional	

Band Pass Filter
 IIR Design — Bilinear Trans.
 Butterworth
 Filter Order: 10
 Filter Design & Analysis
 System
 Momentum Data Systems, Inc.

(a)

Fig. 7.31 Quantization with 16 Bits Fixed Point Fractional



Passband Ripple in -dB	-1.0000
Stopband Ripple in -dB	-30.0000
Passband Cutoff Frequencies	40.0000 82.5000
Stopband Cutoff Frequencies	20.0000 125.000
Sampling Frequency	44100.0
All Frequencies in Hertz	
Quantization 24 Bits Fixed Point Fractional	

Band Pass Filter
IIR Design — Bilinear Trans.
Butterworth
Filter Order: 10
Filter Design & Analysis System
Momentum Data Systems, Inc.

(b)

Fig. 7.31 b) Quantization with 24 Bits Fixed Point Fractional

7.14 REFERENCES

1. Alan V. Oppenheim and Ronald W. Schaffer, *Discrete-Time Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1989.
2. N.K. Bose, *Digital Filters Theory and Applications*, North-Holland, 1985.
3. Johnny R. Johnson, *Introduction to Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1989.
4. Samuel D. Stearns and Ruth A. David, *Signal Processing Algorithms*, Englewood Cliffs, NJ: Prentice Hall, 1988.
5. Harry Y-F. Lam, *Analog and Digital Filters*, Englewood Cliffs, NJ: Prentice Hall, 1979.
6. Chi-Tsong Chen, *One-Dimensional Digital Signal Processing*, New York: Marcel Dekker, 1979.
7. John G. Proakis and Dimitris G. Manolakis, *Introduction to Digital Signal Processing*, New York: Macmillan, 1988.
8. Richard A. Roberts and Clifford T. Mullis, *Digital Signal Processing*, Reading, MA: Addison-Wesley, 1987.

Implementing the Fast Fourier Transform with the DSP56002 Processor

The discrete Fourier transform (DFT) is widely used in digital signal processing applications to determine the frequency content of signals and to perform filtering of signals in the frequency domain. The fast Fourier transform (FFT) is an efficient, high-speed algorithm for computing the discrete Fourier transform (DFT). The FFT achieves its computational high speed by decomposing the data sample set into smaller subsets, in an orderly manner, to eliminate redundant multiplication calculations. Two common methods for decomposing the data sample set are:

1. the decimation-in-time FFT algorithm; and
2. the decimation-in-frequency FFT algorithm.

The DSP56002 processors' instruction set lends itself particularly well to the implementation of the decimation-in-time FFT algorithm. The decimation-in-frequency FFT algorithm can also be implemented on the DSP56002 processor, but the execution speed is somewhat slower than that of the decimation-in-time FFT algorithm.

This chapter begins with a review of the decimation-in-time FFT algorithm. The decimation-in-time FFT algorithm is then implemented on the DSP56002 processor. The DSP56002 assembler macros, used to generate "twiddle" factors, are presented. These macros store the input sequence in bit-reversed order, implement the FFT butterfly, calculate group and coefficient offsets, and perform complete N-point FFTs. The

DSP56002 FFT assembler macros can be found on the DSP56002 Demonstration Software diskette.

8.1 FFT REVIEW

The discrete Fourier transform (DFT) can be defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk} \quad (8.1)$$

where

$$W_N = e^{-j2\pi/N} \quad (8.2)$$

and

$$W_N^{nk} = e^{-j2\pi nk/N} \quad (8.3)$$

For a real or complex discrete-time sequence $x(n)$ of length N , the summation in Equation 8.1 gives a complex frequency $X(k)$ of length N . W_N^{nk} is known as either the twiddle factor or the “weighting” factor. In the remainder of this chapter, N is considered to be an integral power of 2; that is, $N = 2^m$, where m is a positive number.

The FFT can be computed by decomposing the data sample set into smaller subsets, where decomposition is accomplished by using the decimation-in-time FFT algorithm. The idea is to first separate the terms in the DFT summation into two groups: one group containing the terms for which n is even, and the other group containing the terms for which n is odd. Letting $n = 2r$ for the even terms and $n = 2r + 1$ for the odd terms, the DFT can be written in the form:

$$\begin{aligned} X(k) &= \sum_{r=0}^{(N/2)-1} x(2r)W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x(2r+1)W_N^{(2r+1)k} \\ &= \sum_{r=0}^{(N/2)-1} x(2r)W_N^{2rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1)W_N^{2rk} \end{aligned} \quad (8.4)$$

Since N is an even integer and $W_N^2 = W_{N/2} = e^{-j4\pi n/N}$, Equation 8.4 can be rewritten as:

$$X_{1,m}(k) = X_{1,m-1}(k) + W_N^k X_{2,m-1}(k) \quad (8.5)$$

where

$$X_{1,m}(k) = X(k) \quad (8.6)$$

$$X_{1,m-1}(k) = \sum_{r=0}^{(n/2)-1} x(2r)W_{N/2}^{rk} \quad (8.7)$$

$$X_{2,m-1}(k) = \sum_{r=0}^{(n/2)-1} x(2r+1)W_{N/2}^{rk} \quad (8.8)$$

and where $X_{1,m-1}(k)$ and $X_{2,m-1}(k)$ are each an $N/2$ -point DFT.

Since an algorithm is available for computing an $N/2$ -point DFT, $X_{1,m-1}(k)$ and $X_{2,m-1}(k)$ can be combined in accordance with Equation 8.5 and the signal-flow graph shown in Fig. 8.1 to provide the N -point DFT of $x(n)$.

In a similar manner, the two $N/2$ -point DFTs can be written in the form:

$$X_{1,m-1}(k) = X_{1,m-2}(k) + W_N^{2K} X_{2,m-2}(k) \quad (8.9)$$

$$X_{2,m-1}(k) = X_{3,m-2}(k) + W_N^{2K} X_{4,m-2}(k) \quad (8.10)$$

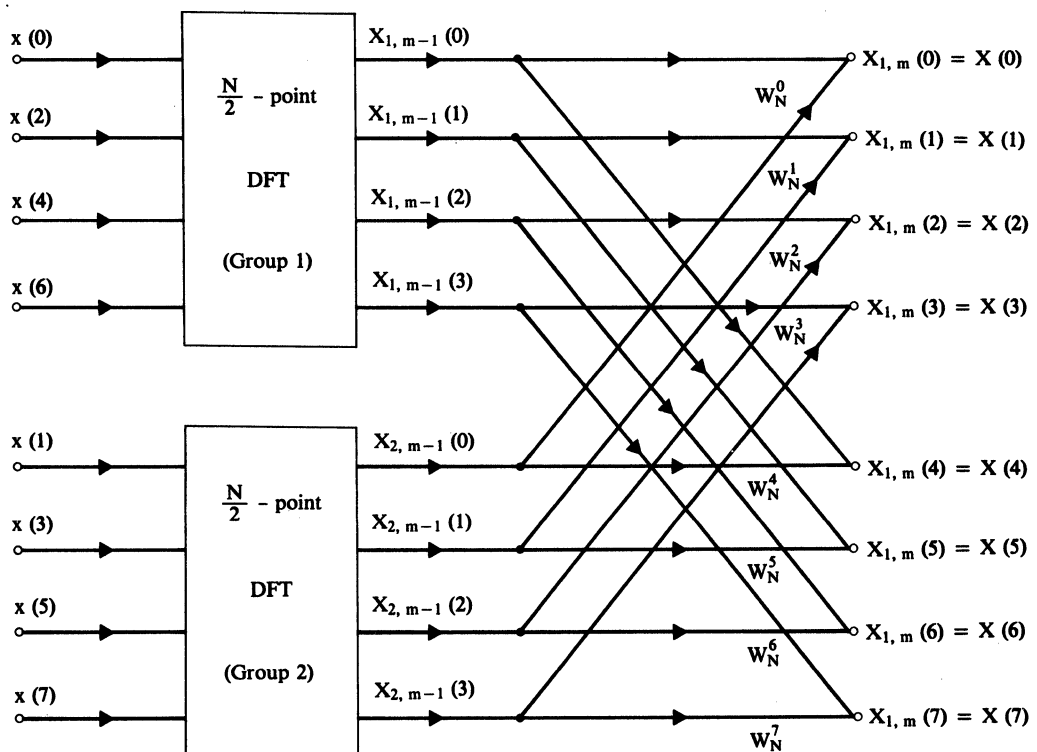


Fig. 8.1 Decimation-in-time Decomposition of N -Point DFT in Terms of Two $N/2$ -point DFTs

where $X_{1,m-2}(k)$, $X_{2,m-2}(k)$, $X_{3,m-2}(k)$ and $X_{4,m-2}(k)$ are each an $N/4$ -point DFT.

Since an algorithm is available for computing an $N/4$ -point DFT, $X_{1,m-2}(k)$, $X_{2,m-2}(k)$, $X_{3,m-2}(k)$, and $X_{4,m-2}(k)$ can be combined in accordance with Equations 8.9, 8.10, and 8.5, and the signal-flow graph shown in Fig. 8.2, to provide an N -point DFT of $x(n)$.

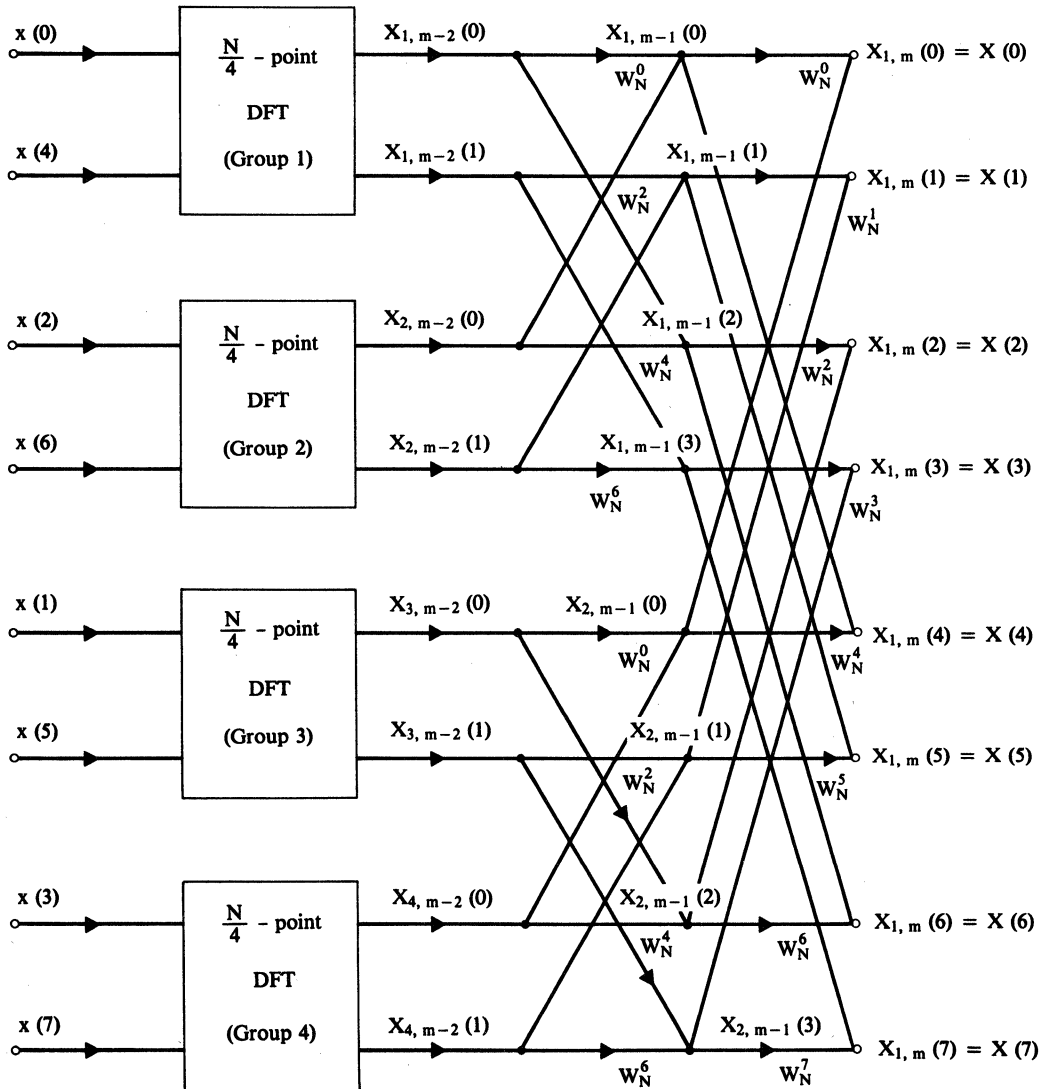


Fig. 8.2 Decomposition of N -point DFT in Terms of $N/4$ -point DFTs

The decomposition process can be continued until a group of $N/2$ two-point (radix-2) DFTs result for the first stage (pass) of the N -point DFT computation, where the total number of stages (passes) m is equal to $\log_2(N)$, e.g., $2^m=N$. The signal-flow graph for $N=8$ input samples, $N/2=4$ groups, and $m=3$ stages is shown in Fig. 8.3, where each solid dot at the intersection of two or more lines indicates a memory location for storage of either an input data sample, the result of an intermediate computation, or a frequency domain output sample.

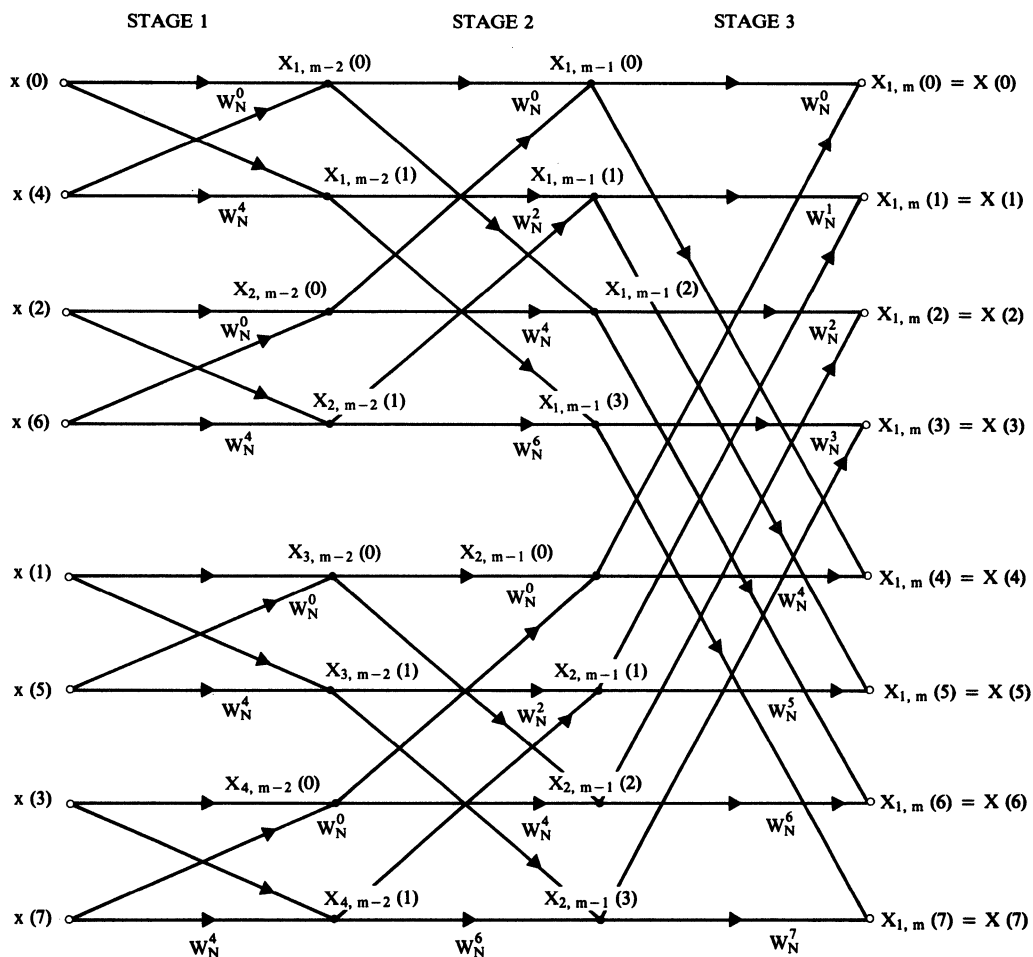


Fig. 8.3 Decimation-in-time Decomposition of N -Point DFT ($N=8$)

8.1.1 Butterfly Computation

The common computation for each group of each stage of the DFT shown in Fig. 8.3 is based on the radix-2 FFT Butterfly shown in Fig. 8.4. Two complex values in one stage are derived from two complex values in the previous stage through the equations:

$$X_{i,j}[p] = X_{i,j-1}[p] + W_N^r X_{i,j-1}[q] \quad (8.11)$$

$$X_{i,j}[q] = X_{i,j-1}[p] + W_N^{r+N/2} X_{i,j-1}[q] \quad (8.12)$$

where i is the group number, j is the stage number, p and q are the RAM data locations for the butterfly.

The values of i , p , q , and r vary from stage to stage as shown in Fig. 8.3. Note that the values of $X_{i,j}[p]$ and $X_{i,j}[q]$ are stored in the same RAM locations as the values of $X_{i,j-1}[p]$ and $X_{i,j-1}[q]$, respectively. Therefore, this type of "in-place" FFT butterfly computation minimizes the total number of RAM locations necessary to compute the DFT.

The coefficients W_N of the butterfly shown in Figure 8.4 are always powers of r or $r+N/2$. Since $W_N^{(r+N/2)} = W_N^r W_N^{N/2} = e^{j\pi} W_N^r = -W_N^r$, the butterfly shown in Fig. 8.4 can be simplified to the butterfly shown in Fig. 8.5, and Equations 8.11 and 8.12 can be rewritten as:

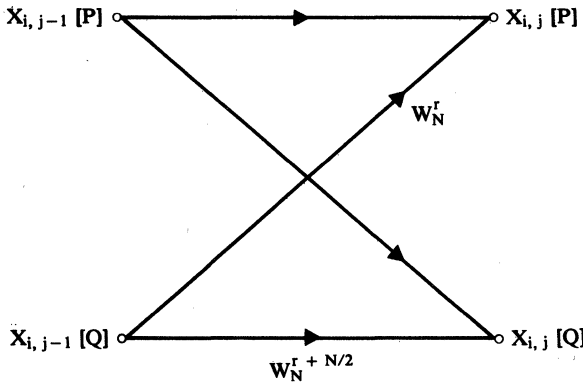


Fig. 8.4 Butterfly Representing Equations 8.11 and 8.12

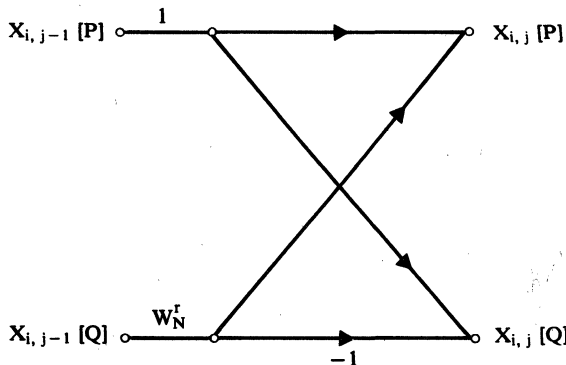


Fig. 8.5 Simplified Butterfly Representing Equations 8.13 and 8.14

$$X_{i,j}[p] = X_{i,j-1}[p] + W_N^r X_{i,j-1}[q] \quad (8.13)$$

$$X_{i,j}[q] = X_{i,j-1}[p] - W_N^r X_{i,j-1}[q] \quad (8.14)$$

The stage number j varies from 1 to m . As shown in Fig. 8.3, each stage j has 2^{m-j} groups and each group i within each stage j has 2^{j-1} butterflies. The distance between the RAM memory locations p and q within a butterfly is referred to as the group offset n_0 , where n_0 is equal to 2^{j-1} . The difference between the values of the coefficient exponents r in group i of stage j is referred to as the coefficient offset n_2 , where n_2 is equal to 2^{m-j} .

$$\begin{aligned} \text{Group offset} \quad n_0 &= 2^{j-1} & j &= 1, 2, \dots, m \\ \text{Coefficient offset} \quad n_2 &= i = 2^{m-j} & j &= 1, 2, \dots, m \end{aligned} \quad (8.15)$$

In computing the DFT, j is incremented from 1 to m , i and n_2 are decremented from 2^{m-1} to 2^0 , and n_0 is incremented from 2^0 to 2^{m-1} .

8.1.2 “Bit-Reversed” Reordering of Input Data

To perform the in-place computations described in section 8.1.1, the elements of the input sequence $x(n)$ must be reordered in a particular manner and stored in consecutive RAM locations. To illustrate the reordering technique, consider each element of the input sequence $x(n)$ to have an associated binary address, where the address of the first input element $x(0)$ is called the base address L and where the m least significant bits of L are assumed to be zero. For an 8-point (eight element) input sequence

$$x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7)$$

the three least significant bits of the associated binary addresses are:

$$000, 001, 010, 011, 100, 101, 110, 111, \text{ respectively.}$$

To reorder the input sequence, the m least significant bits of the address of each sequence element must be “bit-reversed” as shown in Table 8.1.

8.2 RUNNING THE DSP56002 DEMONSTRATION SOFTWARE

To demonstrate the materials in the following sections, you can type the commands that load and execute the FFT programs provided on the DSP56002 Demonstration Software diskette.

8.3 FFT IMPLEMENTATION

The following DSP56002 programs are used to implement the FFT on the DSP56002 processor:

Table 8.1 Input Sequence in Bit-Reversed Order

Input Sequence in Normal Order	LSBs of Address	Bit-Reversed LSBs of Address	Desired Displacement from Base Address L	Input Sequence in Bit-Reversed Order
x(0)	000	000	0	x(0)
x(1)	001	100	4	x(4)
x(2)	010	010	2	x(2)
x(3)	011	110	6	x(6)
x(4)	100	001	1	x(1)
x(5)	101	101	5	x(5)
x(6)	110	011	3	x(3)
x(7)	111	111	7	x(7)

1. A DSP56002 assembler macro to generate twiddle factors is developed in section 8.3.1.
2. A DSP56002 assembler macro to perform bit-reversed reordering of the input sequence x(n) is developed in section 8.3.2.
3. Three different procedures to implement an FFT butterfly on the DSP56002 processor are developed in section 8.3.3.
4. Procedures to compute an FFT's group offset and compute an FFT's coefficient offset using the DSP56002 processor are developed in section 8.3.4.
5. Three different DSP56002 assembler macros that each implement a complete N-point FFT are developed in section 8.3.5. An 8-point FFT example is included.

8.3.1 Generating Twiddle Factors

The twiddle factor in Equation 8.2 can be rewritten as:

$$W_N^k = e^{-j2\pi k/N} = \cos(2\pi k/N) - j\sin(2\pi k/N) \quad (8.16)$$

where $k = 0, 1, \dots, (N - 1)/2$.

The DSP56002 assembler macro **SINCOS.ASM** shown in Fig. 8.6 generates lookup tables for the twiddle factors. The negative of the generated real-part values of Equation 8.16 ($-\cos(2\pi k/N)$) are stored in X-Memory; the generated imaginary-part values ($-\sin(2\pi k/N)$) are stored in Y-Memory. Each table contains one-half cycle of the $-\cos$ and sine waveform, respectively.

The sine and cosine transcendental functions built into the ASM56000 Assembler Software program are used by the assembler macro **SINCOS.ASM** to calculate the sine and cosine table values. Since both sine and cosine values can range from -1 to +1, and since the DSP56002 processor has a fractional-arithmetic range from -1 to

1-2⁻²³ (short of +1), you may choose to avoid using $\cos(0) = +1$ by using the negative of $\cos(0)$ instead.

However, if you do choose to use a cosine waveform instead of a negative cosine waveform, the following assembler warning message will appear on your screen when `cos(0)` is calculated:

Warning—Floating point value outside fractional domain

Since the assembler will substitute the maximum positive fractional data value \$7fffff for $\cos(0)$, and since negligible error is thereby introduced, this warning can be ignored.

Fig. 8.6 SINCOS.ASM Assembler Macro for Generating Sine and Cosine Lookup Tables

```

;*****
;
; Sine-Cosine Table Generator for FFTs
;
;*****
;
;
sincos    macro    points,coef
;
;
;   points    - number of points
;   coef      - base address of sine/cosine table
;               - negative cosine value in X memory
;               - negative sine value in Y memory
;
;
;
pi        equ      3.141592654
freq      equ      2.0*pi/@cvf(points)      ;freq=2pi/points

count     org      x:coef                    ; Calculation of
count     set      0                        ; -cos(freq*count) for
count     dup      points/2                  ; count=0,...,(points/2)-1

count     dc       -@cos(@cvf(count)*freq)
count     set      count+1
count     endm

count     org      y:coef                    ; Calculation of
count     set      0                        ; -sin(freq*count) for
count     dup      points/2                  ; count=0,...,(points/2)-1

count     dc       -@sin(@cvf(count)*freq)
count     set      count+1
count     endm

count     endm                                ; end of sincos macro

```

E X A M P L E

8 - 1

Execute the program SINEX.CLD to generate sine and cosine lookup table values for an 8-Point FFT

In this example, the base addresses for the cosine and sine lookup tables in X-Memory and Y-Memory, respectively, are chosen to be equal to the FFT size $N=8$. Therefore, the table values of $-\cos(2\pi k/8)$ for $k = 0,1,2,3$ are stored in X-Memory locations $\$8-\b ; the table values of $-\sin(2\pi k/8)$ for $k = 0,1,2,3$ are stored in Y-Memory locations $\$8-\b . The SINEX.ASM assembler program is shown in Fig. 8.7.

- (1) Load the Debug-EVM software program.
- (2) Insert the DSP56002 Demonstration Software diskette into Drive "A."
- (3) Select the data1 window and change its radix to fractional (FRA) with a mouse click on the radix button.
- (4) Select the command window and type the EVM commands:


```
load a:sinex<return>
display x:$8..$b<return>
display y:$8..$b<return>
```
- (5) Type quit<return> if you want to exit the Debug-EVM program.

The values in the cosine and sine lookup tables after execution of the SINEX.CLD program are:

```
X:$0008 -1.0000000 -0.7071068  0.0000000  0.7071068
Y:$0008  0.0000000 -0.7071068 -1.0000000 -0.7071068
```

Fig. 8.7 SINEX.ASM Assembler Program

```
;
; Twiddle factors for 8-point FFT
;
;
points      equ          $8
coef        equ          $8

; points - number of points
; coef   - base address of sine/cosine table
; negative cosine value in X memory
; negative sine value in Y memory
;
;

        include 'sincos'
sincos          points,coef
end
```

8.3.2 Storing the Input Sequence $X(n)$ in Bit-Reversed Order

The DSP56002 processor's Address Generation Unit (AGU) has an addressing mode that automatically performs reverse-carry (bit-reversed) arithmetic. When using this addressing mode in conjunction with computing an FFT, the DSP56002 pro-

cessor will automatically select the appropriate input (or output) sequence element, thereby enhancing the FFT's execution speed by making storing of the reordered input (or output) sequence unnecessary. In order to fully illustrate the reverse-carry arithmetic concept as a subject in itself, the remainder of this section uses the DSP56002 processor's AGU to first reorder the input sequence $x(n)$ and then store the reordered results. Reverse-carry arithmetic can be used to perform a bit-reversed modification of the m least significant bits of a memory address. This can be accomplished by adding an offset $N=2^{m-1}$ to the address value and then propagating any generated carry term in the reverse direction; e.g., the carry term propagates in the direction of the least significant bit.

The BITREV.ASM assembler macro shown in Fig. 8.8 reorders an input sequence $x(n)$ by performing bit-reversed address modifications as described above, and then stores the reordered results. The input sequence is assumed to be originally in normal order, with its real-part values stored in X-Memory locations L through $L+N-1$, and its imaginary-part values stored in Y-Memory locations L through $L+N-1$, where L is the base address of both. The m least significant bits of L must be zeros. After execution of the assembler macro, the reordered input sequence's real-part values are stored in X-Memory locations 0 through $N-1$ and its imaginary-part values are stored in Y-Memory locations 0 to $N-1$. Since the sine table is stored in X-Memory locations N through $3N/2-1$ and the cosine table is stored in Y-Memory locations N through $3N/2-1$, then L can be greater than or equal to $3N/2$.

The BITREV.ASM assembler macro uses the DSP56002 processor's ability to postincrement an address by an offset N with reverse-carry arithmetic, thereby performing a bit-reversed modification on the address. The base addresses L of the normally ordered elements of the input sequence are initially stored in AGU Registers $R1$ and $R6$, where $R1$ points to X-Memory and $R6$ points to Y-Memory; the base addresses of the reordered elements of the output sequence are initially stored in $R0$ and $R5$, where $R0$ points to X-Memory and $R5$ points to Y-Memory. $R1$ and $R6$ are each postincremented by an offset N with reverse-carry arithmetic. Registers $N1$ and $N6$ are programmed with the offset value $N=2^{m-1}$; Registers $M1$ and $M6$ are programmed with the value $\$ffff$ to select the reverse-carry arithmetic mode. $R0$ and $R5$ are each postincremented by one with linear arithmetic, where linear arithmetic is selected by programming $M0$ and $M5$ with the value $\$ffff$. The assembler syntax itself defines the postincrement by one condition, consequently $N0$ and $N5$ need not be programmed. Table 8.2 shows the reverse-carry address modifications for input sequence elements $0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7$, where $N=8$ and $L=\$18$.

Fig. 8.8 BITREV.ASM Assembler Macro for Storing Input Sequence in Bit-Reversed Order

```
;
;*****
; Storing Input Data in Bit-Reversed Order
;
;*****
bitrev macro points,data,olddata
;
```

```

;   points = Number of points
;   olddata = Input data in normal order
;   data    = Input data in bit-reversed order
;
;
;   move    #olddata,r1      ; initialize the
;   move    r1,r6            ; pointers
;   move    #data,r0
;   move    r0,r5
;
;   move    #0,m1            ; m1 and m6 = bit
;   move    m1,m6            ; reverse
;
;   move    #-1,m0           ; m5 and m6 = linear
;   move    m0,m5
;
;   move    #points/2,n1
;   move    n1,n6
;
;   do      #points,_end_reverse
;   move    x:(r1)+n1,a      y:(r6)+n6,b
;   move    a,x:(r0)+       b,y:(r5)+
;
_end_reverse
endm

```

Table 8.2 Address Modifications for N=8 and L=\$18

Input Sequence in Normal Order	Address Modification	Input Sequence in Bit-Reversed Order
X:\$18 0.0	00011000 = 18 + 1	X:\$0 0.0
X:\$19 0.1	00011100 = 1C + 1	X:\$1 0.4
X:\$1A 0.2	00011010 = 1A + 1	X:\$2 0.2
X:\$1B 0.3	00011110 = 1E + 1	X:\$3 0.6
X:\$1C 0.4	00011001 = 19 + 1	X:\$4 0.1
X:\$1D 0.5	00011101 = 1D + 1	X:\$5 0.5
X:\$1E 0.6	00011011 = 1B + 1	X:\$6 0.3
X:\$1F 0.7	00011111 = 1F	X:\$7 0.7

E X A M P L E

8 – 2

Execute the program BITEV.CLD to store an 8-point input sequence in bit-reversed order

In this example, the original input sequence is in normal order with its real-part values stored in X-Memory locations \$18-\$1f and its imaginary-part values stored in Y-Memory locations \$18-\$1f. After execution of the program **BITEV.CLD**, input sequence is in bit-reversed order with its real-part values stored in X-Memory locations \$0-\$7 and its imaginary-part values stored in Y-Memory locations \$0-\$7. The **BITEV.ASM** assembler program is shown in Fig. 8.9. Note that it has an “include” statement for the **BITREV.ASM** assembler program shown in Fig. 8.8.

- (1) Load the Debug-EVM Software program.
- (2) Insert the DSP56002 Demonstration Software diskette into Drive “A.”
- (3) Select the data1 window and change its radix to fractional (FRA) with a mouse click on the radix button.
- (4) Select the command window and type the EVM commands:

```
load a:bitex<return>
break #1 p:$110<return>
go <return>
display x:$18..$1f<return>
display y:$18..$1f<return>
display x:$0..$7<return>
display y:$0..$7<return>
```

Input sequence in normal order:

Real parts:

```
X:$0018 0.0000000 0.1000000 0.2000000 0.3000000
X:$001C 0.4000000 0.5000000 0.6000000 0.7000000
```

Imaginary parts:

```
Y:$0018 0.0000000 0.1000000 0.2000000 0.3000000
Y:$001C 0.4000000 0.5000000 0.6000000 0.7000000
```

Input sequence in bit-reversed order:

Real parts:

```
X:$0000 0.0000000 0.4000000 0.2000000 0.6000000
X:$0004 0.1000000 0.5000000 0.3000000 0.7000000
```

Imaginary parts:

```
Y:$0000 0.0000000 0.4000000 0.2000000 0.6000000
Y:$0004 0.1000000 0.5000000 0.3000000 0.7000000
```

- (5) Type quit<return> if you want to exit the Debug-EVM program.

Fig. 8.9 BITEX.ASM Assembler Program

```

;
;*****
;
; 8-point Bit-Reversed Example
;
;*****
;
;       include 'bitrev'
;   Input Data
;
;   points = Number of points
;   olddata = Input data in normal order
;   data    = Input data in bit-reversed order
;
data      equ      $0              ; Begin of data
points    equ      $8              ; FFT size
olddata   equ      $18
;
;       org      x:$18              ; Begin of olddata
;       ; (real part)
;       dc      0.0,0.1,0.2,0.3
;       dc      0.4,0.5,0.6,0.7
;
;       org      y:$18              ; Begin of olddata
;       ; (imaginary part)
;       dc      0.0,0.1,0.2,0.3
;       dc      0.4,0.5,0.6,0.7
;
; program start
;
;       org      p:$100
;
;       bitrev    points,data,olddata
;
;       end

```

8.3.3 Butterfly Computation

Equations 8.13 and 8.14 are used to compute the butterfly shown in Fig. 8.5. If the real- and imaginary-parts of $X_{i,j-1}[p]$, $X_{i,j-1}[q]$, $X_{i,j}[p]$, $X_{i,j}[q]$ and W_N^r are:

$$\begin{aligned}
 X_{i,j-1}[p] &= ar + jai \\
 X_{i,j-1}[q] &= br + jbi \\
 X_{i,j}[p] &= ar' + jai' \\
 X_{i,j}[q] &= br' + jbi' \\
 W_N^r &= wr + jwi
 \end{aligned}
 \tag{8.17}$$

then the butterflies' outputs ar' , ai' , br' , and bi' can be calculated from Equations 8.13, 8.14, and 8.17 as:

$$\begin{aligned}
ar' &= ar + wr*br - wi*bi \\
ai' &= ai + wi*br + wr*bi \\
br' &= ar - wr*br + wi*bi = 2*ar - ar' \\
bi' &= ai - wi*br - wr*bi = 2*ai - ai'
\end{aligned} \tag{8.18}$$

where after computation of the butterfly, ar' and br' are stored in X-Memory and ai' and bi' are stored in Y-Memory.

Since the twiddle factor used in Equations 8.13 and 8.14 has the form $e^{-j2\pi r/N}$, which has unit magnitude, the magnitude of $X_{i,j}[p]$ and $X_{i,j}[q]$ can grow by a maximum factor of two per stage. For an N-point FFT, the magnitude can grow by a total factor of N. Consequently, the magnitude of $X_{i,j}[p]$, and $X_{i,j}[q]$, must be kept to less than one such that they can be stored without overflow or limiting. This can be accomplished by using one of the following three procedures:

1. Limit the magnitude of the input sequence elements to $1/N$, where the magnitude is calculated as the square root of the sum of the squares of the real-part and imaginary-part values.
2. Automatically down-scale by a factor of 0.5 the outputs of all of the butterflies within the FFT. The magnitude of the FFT input data is less than one. A common scale factor of N needs to be applied to the FFT results if the absolute (nonscaled) results are required. The DSP56002 processor can be selected to automatically down-scale data by a factor of 0.5. This is accomplished by programming bits 10 and 11 of its Status Register to 0 and 1, respectively.
3. Use a block floating point technique to accomplish conditional scaling. This technique allows complete FFT passes (stages) to be selectively scaled depending on the growth in magnitude that occurred in the previous pass. This technique can be implemented most efficiently by using the Scaling Bit S in the Status Register (bit 7). The Scaling Bit is set when the DSP56002 processor moves a result larger than 0.25 in absolute value to memory. The FFT algorithm tests the Scaling Bit upon completion of each FFT pass. If it is set, the DSP56002 processor's automatic scaling mode is turned on before the next pass. If the Scaling Bit is not set, no growth has occurred and the automatic scaling mode is turned off. If absolute (nonscaled) values are required, then a scaling factor needs to be calculated. The scaling factor is equal to 2^k , where k is the number of passes for which automatic down-scaling was turned on.

E X A M P L E

8 – 3

Limit to $1/N = 0.5$ the magnitude of the input values of a single 2-point butterfly

In this example, the **BUTER1.CLD** program is executed to compute the output values ar' , ai' , br' , and bi' of a single 2-point butterfly. The real and imaginary parts of the input values are: $ar = 0.4$, $ai = 0.25$, $br = 0.1$, and $bi = 0.15$; the real and imaginary parts of the twiddle factor values are $wr = -1$ and $wi = 0$. The real-part input values ar and br are stored in X-Memory locations \$0 and \$1, respectively; the imaginary-part input values ai and bi are stored in Y-Memory locations \$0 and \$1, respectively. The real-part value of the

twiddle factor is stored in X-Memory location \$2; the imaginary-part value of the twiddle factor is stored in the Y-Memory location \$2.

After the execution of the **BUTER1.CLD** program, the real-part output values *ar'* and *br'* are stored in X-Memory locations \$0 and \$1, respectively; the imaginary-part output values *ai'* and *bi'* are stored in Y-Memory locations \$0 and \$1. Note that since the output values overwrite the input values in both X-Memory and Y-Memory, the **BUTER1.CLD** program is an in-place butterfly algorithm. The BUTER1.ASM assembler program is shown in Fig. 8.10.

- (1) Load the Debug-EVM Software program.
- (2) Insert the DSP56002 Demonstration Software diskette into Drive "A."
- (3) Select the data1 window and change its radix to fractional with a mouse click on the radix button.
- (4) Select the command window, change its radix to fractional, and type the EVM commands:

```
change x:$0 0.4 y:$0 0.25<return>
change x:$1 0.1 y:$1 0.15<return>
change x:$2 -1.0 y:$2 0<return>
display x:$0..$1 <return>
display y:$0..$1<return>
load a:buter1<return>
break #`1 p:$11e<return>
go<return>
display x:$0..$1 <return>
display y:$0..$1<return>
```

- (5) Type quit<return> if you want to exit the Debug-EVM program.

Butterfly inputs:

```
X:$0000 0.4000000 0.1000000
Y:$0000 0.2500000 0.1500000
```

Butterfly outputs:

```
X:$0000 0.5000000 0.3000000
Y:$0000 0.4000000 0.1000000
```

Fig. 8.10 BUTER1.ASM Assembler Program

```
;*****
; 2-point FFT
;
;*****

passes      equ      $1
points      equ      $2
data        equ      $0
coef        equ      $2

                org      p:$100
```

```

        move      #1,n0                ; n0 = group offset
        move      #points/2,n2        ; n2 = group per pass

        move      #-1,m0               ; m0=m1=m2=m4=m5=m6
        move      m0,m1                ; = linear addressing
        move      m0,m2
        move      m0,m4
        move      m0,m5
        move      m0,m6

        move      n0,n1                ; n0 = n1 = n4 = n5
        move      n0,n4                ; group offset
        move      n0,n5
        move      n2,n6                ; n6 = coefficient offset

        move      #data,r0             ;r0 = ar,ai input pointer
        move      r0,r4                ;r4 = ar',ai' output pointer
        move      #coef,r6             ;r6 = wr,wi input pointer

        nop
        lua       (r0)+n0,r1           ;r1 = br,bi input pointer
        nop
        move      r1,r5                ;r5 = br',bi' input pointer

;*****
;
;   2-point butterfly
;
;*****

        move      y:(r0),b             ;b = ai
        move      x:(r1),x1            y:(r6),y0        ;x1 = br
                                                ;y0 = wi

        mac  x1,y0,b    x:(r6),x0      y:(r1),y1        ;b = ai+br*wi
                                                ;x0 = -wr
                                                ;y1 = bi

        macr -x0,y1,b                y:(r0),a           ;b = ai+br*wi+bi*wr
                                                ; = ai'
                                                ;a = ai

        subl b,a    x:(r0),b    b,y:(r4)                ;a = 2ai-ai' = bi'
                                                ;b = ar
                                                ;y(r4) = ai'

        mac -x0,x1,b    x:(r0),a    a,y:(r5)            ;b = ar+br*wr
                                                ;a = ar
                                                ;y(r5) = bi'

        macr -y0,y1,b                ;b = ar+br*wr-bi*wi
                                                ; = ar'

        subl b,a    b,x:(r4)          ;a = 2ar-ar' = br'
                                                ;x:(r4) = ar'

        move      a,x:(r5)            ;x:(r5) = br'
        end

```

E X A M P L E

8 - 4

Automatically down-scale by a factor of 0.5 the outputs of a single 2-point butterfly

In this example, the **BUTER2.CLD** program is executed to compute the output values ar', ai', br', and bi' of a 2-point butterfly, where the magnitudes of its input values are limited to be less than 1.0 and its output values are halved. The real and imaginary parts of the input values are: ar = 0.8, ai = 0.5, br = 0.2, and bi = 0.3; the real and imaginary parts of the twiddle factor values are wr = -1 and wi = 0. The real-part input values ar and br are stored in X-Memory locations \$0 and \$1, respectively; the imaginary-part input values ai and bi are stored in Y-Memory locations \$0 and \$1, respectively. The real-part value of the twiddle factor is stored in X-Memory location \$2; the imaginary-part value of the twiddle factor is stored in the Y-Memory location \$2.

After the execution of the **BUTER2.CLD** program, the real-part output values ar' and br' are stored in X-Memory locations \$0 and \$1, respectively; the imaginary-part output values ai' and bi' are stored in Y-Memory locations \$0 and \$1. Note that since the output values overwrite the input values in both X-Memory and Y-Memory, the **BUTER2.CLD** program is an in-place butterfly algorithm. The **BUTER2.ASM** program is shown in Fig. 8.11. Note that use of the ORI instruction enables the down-scaling mode and use of the ANDI instruction disables down-scaling mode.

- (1) Load the Debug-EVM Software program.
- (2) Insert the DSP56002 Demonstration Software diskette into Drive "A."
- (3) Select the data1 window and change its radix to fractional with a mouse click on the radix button.
- (4) Select the command window, change its radix to fractional, and type the EVM commands:

```
change x:$0 0.8 y:$0 0.5<return>
change x:$1 0.2 y:$1 0.3<return>
change x:$2 -1.0 y:$2 0<return>
display x:$0..$1 <return>
display y:$0..$1<return>
load a:buter2<return>
break #1 p:$120<return>
go<return>
display x:$0..$1 <return>
display y:$0..$1<return>
```

- (5) Type quit<return> if you want to exit the Debug-EVM program.

Butterfly inputs:

```
X:$0000 0.8000000 0.2000000
Y:$0000 0.5000000 0.3000000
```

Butterfly outputs:

```
X:$0000 0.5000000 0.3000000
Y:$0000 0.4000000 0.1000000
```

Fig. 8.11 Buter2.ASM Assembler Program

```

;*****
;   2-point FFT
;
;*****

passes      equ      $1
points      equ      $2
data        equ      $0
coef        equ      $2

        org      p:$100

        move     #1,n0          ; n0 = group offset
        move     #points/2,n2    ; n2 = group per pass

        move     #-1,m0          ; m0=m1=m2=m4=m5=m6
        move     m0,m1          ; = linear addressing
        move     m0,m2
        move     m0,m4
        move     m0,m5
        move     m0,m6

        move     n0,n1          ; n0 = n1 = n4 = n5
        move     n0,n4          ; group offset
        move     n0,n5
        move     n2,n6          ; n6 = coefficient offset

        move     #data,r0       ; r0 = ar,ai input pointer
        move     r0,r4          ; r4 = ar',ai'output pointer
        move     #coef,r6       ; r6 = wr,wi input pointer

        nop
        lua      (r0)+n0,r1     ; r1 = br,bi input pointer
        nop
        move     r1,r5          ; r5 = br',bi' input pointer

;*****
;
;   2-point butterfly
;
;*****

        ori      #$4,mr

        move     y:(r0),b       ; b = ai
        move     x:(r1),x1      y:(r6),y0    ; x1 = br
                                           ; y0 = wi

        mac      x1,y0,b        x:(r6),x0      y:(r1),y1    ; b = ai+br*wi
                                           ; x0 = -wr
                                           ; y1 = bi

        macr     -x0,y1,b       y:(r0),a        ; b = ai+br*wi+bi*wr
                                           ; = ai'
                                           ; a = ai

```

```

subl    b,a          x:(r0),b      b,y:(r4)      ;a = 2ai-ai' = bi'
                                           ;b = ar
                                           ;y(r4) = ai'

mac     -x0,x1,b      x:(r0),a      a,y:(r5)      ;b = ar+br*wr
                                           ;a = ar
                                           ;y(r5) = bi'

macr    -y0,y1,b                        ;b = ar+br*wr-bi*wi
                                           ; = ar'

subl    b,a          b,x:(r4)      ;a = 2ar-ar' = br'
                                           ;x:(r4) = ar'

move    a,x:(r5)      ;x:(r5) = br'

andi    #$fb,mr

end

```

E X A M P L E

8 - 5

Use a block floating point technique to perform conditional scaling of the output values of a single 2-point butterfly

Note: This example is not executed using the DSP56002EVM. It is executed using the Motorola's SIM56000 Software Program.

In this example, the **BUTER3.CLD** program is executed to compute the butterfly outputs ar',ai',br',and bi' of a 2-point butterfly, where the magnitudes of its input values are limited to $1/N = 0.5$ and the block floating point technique is used to accomplish conditional scaling of its output values. The real and imaginary parts of the input values are: ar = 0.8, ai = 0.5, br = 0.2, and bi = 0.3; the real and imaginary parts of the twiddle factor values are wr = -1 and wi = 0. The real-part input values ar and br are stored in X-Memory locations \$0 and \$1, respectively; the imaginary-part input values ai and bi are stored in Y-Memory locations \$0 and \$1, respectively. The real-part value of the twiddle factor is stored in X-Memory location \$2; the imaginary-part value of the twiddle factor is stored in the Y-Memory location \$2.

The **BUTER3.CLD** program is executed two times: the first time with the Scaling Bit set to down-scale the butterfly output values by a factor of 0.5, and the second time with the Scaling Bit cleared to obtain the butterfly output values without scaling. After execution of the **BUTER3.CLD** program, the real-part output values ar' and br' are stored in X-Memory locations \$0 and \$1, respectively; the imaginary-part output values ai' and bi' are stored in Y-Memory locations \$0 and \$1. Note that since the output values overwrite the input values in both X-Memory and Y-Memory, the **BUTER3.CLD** program is an in-place butterfly algorithm. The **BUTER3.ASM** program is shown in Fig. 8.12.

- (1) Load the SIM56000 Simulator Software program.
- (2) Insert the DSP56000 Demonstration Software diskette into Drive "A."
- (3) Type the SIM56000 commands:
 - (a) Scaling bit is set:


```
change sr $80<return>
change x:0 `0.4 y:0 `0.25<return>
```

```

change x:1 `0.1 y:1 `0.15<return>
change x:2 -`1.0 y:2 `0<return>
display x:0..1 <return>
display y:0..1<return>
load a:buter3<return>
break #1 pc>$127<return>
go<return>
display r7<return>
display x:0..1 <return>
display y:0..1<return>

```

Butterfly inputs:

X:\$0000 0.4000000 0.1000000

Y:\$0000 0.2500000 0.1500000

Scaling factor: 2^k , where k is equal to 1.

r7= \$0001

Butterfly outputs:

X:\$0000 0.2500000 0.1500000

Y:\$0000 0.2000000 0.0500000

(b) Scaling bit is cleared:

```

change sr $00<return>
change x:0 `0.4 y:0 `0.25<return>
change x:1 `0.1 y:1 `0.15<return>
change x:2 -`1.0 y:2 `0<return>
display x:0..1<return>
display y:0..1<return>
load a:buter3<return>
break #1 pc>$127<return>
go<return>
display r7<return>
display x:0..1<return>
display y:0..1<return>

```

Butterfly inputs:

X:\$0000 0.4000000 0.1000000

Y:\$0000 0.2500000 0.1500000

Scaling factor: 2^k , where k is equal to 0.

r7= \$0000

Butterfly outputs:

X:\$0000 0.5000000 0.3000000

Y:\$0000 0.4000000 0.1000000

(4) Type quit<return> if you want to exit the SIM56000 program.

Fig. 8.12 Buter3.ASM Assembler Program

```

;*****
; 2-point FFT
;
;*****

passes      equ      $1
points      equ      $2
data        equ      $0
coef        equ      $2
accr        equ      $3

                org      p:$100

                move     #1,n0          ; n0 = group offset
                move     #points/2,n2    ; n2 = group per pass

                move     #-1,m0          ; m0=m1=m2=m4=m5=m6
                move     m0,m1          ; = linear addressing
                move     m0,m2
                move     m0,m4
                move     m0,m5
                move     m0,m6

                move     #accr,r3
                move     n0,n1          ; n0 = n1 = n4 = n5
                move     n0,n4          ; group offset
                move     n0,n5
                move     n2,n6          ; n6 = coefficient offset

                move     #data,r0        ;r0 = ar,ai input pointer
                move     r0,r4          ;r4 = ar',ai'output pointer
                move     #coef,r6       ;r6 = wr,wi input pointer

                nop
                lua      (r0)+n0,r1     ;r1 = br,bi input pointer
                nop
                move     r1,r5          ;r5 = br',bi' input pointer

                move     #$0,r7

;*****
;
; 2-point butterfly
;
;*****

                move     sr,x:(r3)
                jclr     #7,x:(r3),noscale
                ori      #$4,mr
                move     (r7)+

noscale
                move     y:(r0),b      ;b = ai

                andi     #$7f,ccr

                move     x:(r1),x1     y:(r6),y0      ;x1 = br

```

```

mac      x1,y0,b      x:(r6),x0      y:(r1),y1      ;y0 = wi
                                                ;b = ai+br*wi
                                                ;x0 = -wr
                                                ;y1 = bi
macr     -x0,y1,b      y:(r0),a      ;b = ai+br*wi+bi*wr
                                                ; = ai'
                                                ;a = ai
subl     b,a          x:(r0),b      b,y:(r4)      ;a = 2ai-ai' = bi'
                                                ;b = ar
                                                ;y(r4) = ai'
mac      -x0,x1,b      x:(r0),a      a,y:(r5)      ;b = ar+br*wr
                                                ;a = ar
                                                ;y(r5) = bi'
macr     -y0,y1,b      ;b = ar+br*wr-bi*wi
                                                ; = ar'
subl     b,a          b,x:(r4)      ;a = 2ar-ar' = br'
                                                ;x(r4) = ar'
move     a,x:(r5)      ;x(r5) = br'

end

```

8.3.4 Group and Coefficient Offsets

The **PASSEX.ASM** assembler program shown in Fig. 8.13 computes the group offset n_0 and the coefficient offset n_2 , where n_0 and n_2 are defined by Equation 8.15.

E X A M P L E 8 – 6

In this example, the **PASSEX.CLD** program is used to compute the group and coefficient offsets for the three stages of an 8-point FFT. The **PASSEX.ASM** assembler program is shown in Fig. 8.13.

- (1) Load the Debug-EVM Software program.
- (2) Insert the DSP56002 Demonstration Software diskette into Drive “A.”
- (3) Select the data1 window and change its radix to fractional, with a mouse click on the radix button.
- (4) Select the command window, change its radix to fractional, and type the EVM commands:

```

load a:passex<return>
step `2<return>
step `7<return>
step `4<return>

```

- (5) Type quit<return> if you want to exit the Debug-EVM program.

The value of the group offset n_0 and the coefficient offset n_2 for each of the three stages of the FFT are:

First stage	$n_0 = 1$	$n_2 = 4$
Second Stage	$n_0 = 2$	$n_2 = 2$
Third Stage	$n_0 = 4$	$n_2 = 1$

Fig. 8.13 PASSEX.ASM Assembler Program

```

;
;
;   passes = number of passes
;   points = FFT size
;
passes      equ      $3
points      equ      $8

      move      #1,n0      ; n0 = 1,2,4
      move      #points/2,n2      ; n2 = 4,2,1

      move      #-1,m0      ; m0=m1= linear
      move      m0,m2

      do        #passes,_end_pass

      move      n2,b1
      lsr       b      n0,a1
      lsl       a      b1,n2
      move      a1,n0

_end_pass

      end

```

8.3.5 FFT Macros

The FFT macros **FFT1.ASM**, **FFT2.ASM**, and **FFT3.ASM** can each perform complete FFTs on a complex input sequence. The three macros are shown in Figs. 8.14, 8.15, and 8.16, respectively. The macros can be called with the following arguments:

1. number of FFT points
2. location of the normally ordered or bit-reverse ordered input sequence
3. location of the sine-cosine table

All DSP56002 register initialization is performed within the macros, and the output sequence is in normal order. In the **FFT1** macro, the elements of the input sequence are assumed to be limited to a magnitude of $1/N$. In the **FFT2** macro, the outputs of all of the FFT butterflies are down-scaled by a factor of two. In the **FFT3** macro, the elements of the input sequence are limited to a magnitude of 0.5, the block floating point technique is used to accomplish conditional scaling, and a memory location is used to store the contents of the DSP56002 processor's Status Register.

Fig. 8.14 FFT1.ASM Assembler Macro

```

;*****
;
;   Decimation in Time FFT

```

```

;
; *****
fft1 macro points,data,coef,olddata
; passes = number of passes = log2 ( number of points)
; points = number of points
; coef = base address of sine/cosine table
; olddata = Starting address of input data before bit reverse
; data = Starting address of input data after bit reverse
;
;
; *****
; FFT Initialization
;
; *****
;
;         move    #1,n0           ; n0 = group offset
;         move    #points/2,n2     ; n2 = group per pass
;         move    #-1,m0           ; m0=m1=m2=m4=m5=m6
;         move    m0,m1           ; = linear addressing
;         move    m0,m2
;         move    m0,m4
;         move    m0,m5
;         move    m0,m6
;
; *****
;
;         FFT passes
;
; *****
;
;         do      #@cvi(@log(points)/@log(2)+0.5),_end_pass
;
;         move    #data,r0         ; r0 = ar,ai input pointer
;         move    r0,r4            ; r4 = ar',ai' output pointer
;         move    #coef,r6         ; r6 = wr,wi input pointer
;
;         nop
;         lua     (r0)+n0,r1       ; r1 = br,bi input pointer
;         nop
;         lua     (r1)-,r5         ; r5 = br',bi' input pointer
;
;         move    n0,n1           ; n0 = n1 = n4 = n5
;         move    n0,n4           ; group offset
;         move    n0,n5
;         move    n2,n6           ; n6 = coefficient offset
;
; *****
;
;         FFT Group
;
; *****
;
;         do      n2,_end_grp
;
;         move    x:(r5),a         y:(r0),b

```

```

;*****
;
;   FFT Butterfly
;
;*****

    do    n0,_end_bfy

    move          x:(r1),x1          y:(r6),y0
    mac          x1,y0,b          x:(r6)+n6,x0          y:(r1)+,y1
    macr          -x0,y1,b          a,x:(r5)+          y:(r0),a
    subl         b,a          x:(r0),b          b,y:(r4)
    mac          -x0,x1,b          x:(r0)+,a          a,y:(r5)
    macr          -y0,y1,b          x:(r1),x1
    subl         b,a          b,x:(r4)+          y:(r0),b

_end_bfy

    move          #coef,r6
    move          a,x:(r5)+n5          y:(r1)+n1,y1
    move          x:(r0)+n0,x1          y:(r4)+n4,y1

_end_grp

    move          n2,b1
    lsr          b          n0,a1
    lsl          a          b1,n2
    move          a1,n0

_end_pass

    endm

```

Fig. 8.15 FFT2.ASM Assembler Macro

```

;*****
;
;   Decimation in Time FFT
;
;*****

fft2    macro points,data,coef,olddata

;   passes = number of passes = log2 ( number of points)
;   points = number of points
;   coef   = base address of sine/cosine table
;   olddata = Starting address of input data before bit reverse
;   data    = Starting address of input data after bit reverse

;
;
;*****
;
;   FFT Initialization
;
;*****

    move          #1,n0          ; n0 = group offset

```

```

        move      #points/2,n2      ; n2 = group per pass

        move      #-1,m0            ; m0=m1=m2=m4=m5=m6
        move      m0,m1              ; = linear addressing
        move      m0,m2
        move      m0,m4
        move      m0,m5
        move      m0,m6

; *****
;
;      FFT passes
;
; *****

        do        #@cvi(@log(points)/@log(2)+0.5),_end_pass

        move      #data,r0          ; r0 = ar,ai input pointer
        move      r0,r4              ; r4 = ar',ai' output pointer
        move      #coef,r6          ; r6 = wr,wi input pointer

        nop
        lua       (r0)+n0,r1        ; r1 = br,bi input pointer
        nop
        lua       (r1)-,r5          ; r5 = br',bi' input pointer

        move      n0,n1              ; n0 = n1 = n4 = n5
        move      n0,n4              ; group offset
        move      n0,n5
        move      n2,n6              ; n6 = coefficient offset

; *****
;
;      FFT Group
;
; *****

        ori       #$4,mr

        do        n2,_end_grp

        move      x:(r5),a          y:(r0),b
        lsl       a

; *****
;
;      FFT Butterfly
;
; *****

        do        n0,_end_bfy

        move      x:(r1),x1          y:(r6),y0
        mac       x1,y0,b            x:(r6)+n6,x0    y:(r1)+,y1
        macr      -x0,y1,b          a,x:(r5)+        y:(r0),a
        subl     b,a                x:(r0),b          b,y:(r4)
        mac       -x0,x1,b          x:(r0)+,a        a,y:(r5)
        macr      -y0,y1,b          x:(r1),x1
        subl     b,a                b,x:(r4)+        y:(r0),b

```

```

_end_bfy
    move    #coef,r6
    move    a,x:(r5)+n5      y:(r1)+n1,y1
    move    x:(r0)+n0,x1     y:(r4)+n4,y1

_end_grp
    andi    $$fb,mr

    move    n2,b1
    lsr     b      n0,a1
    lsl     a      b1,n2
    move    a1,n0

_end_pass
    endm

```

Fig. 8.16 FFT3.ASM Assembler Macro

```

;*****
;
;   Decimation in Time  FFT
;
;*****

fft3    macro points,data,coef,olddata,accr

;   passes = number of passes = log2 ( number of points)
;   points = number of points
;   coef   = base address of sine/cosine table
;   olddata = Starting address of input data before bit reverse
;   data    = starting address of input data after bit reverse
;   accr    = address used to store the value of the ccr register
;
;*****
;
;   FFT Initialization
;
;*****

    move    #0,r7
    move    #1,n0      ; n0 = group offset
    move    #points/2,n2 ; n2 = group per pass

    move    #-1,m0      ; m0=m1=m2=m4=m5=m6
    move    m0,m1      ; = linear addressing
    move    m0,m2
    move    m0,m4
    move    m0,m5
    move    m0,m6
    move    m0,m7

;*****
;
;   FFT passes

```

```

;
;*****
do      #@cvi(@log(points)/@log(2)+0.5),_end_pass

move    #data,r0      ;r0 = ar,ai input pointer
move    r0,r4         ;r4 = ar',ai'output pointer
move    #coef,r6      ;r6 = wr,wi input pointer

nop
lua      (r0)+n0,r1    ;r1 = br,bi input pointer
nop
lua      (r1)-,r5      ; r5 = br',bi' input pointer

move    #accr,r3

move    n0,n1         ; n0 = n1 = n4 = n5
move    n0,n4         ; group offset
move    n0,n5
move    n2,n6         ; n6 = coefficient offset
;*****
;
;   FFT Group
;
;*****

move    sr,x:(r3)
jclr    #7,x:(r3),noscale
ori     #$4,mr
move    (r7)+
noscale

do      n2,_end_grp
move    x:(r5),a      y:(r0),b

jclr    #7,x:(r3),continue
lsl     a
continue

andi    #$7f,ccr
;*****
;
;   FFT Butterfly
;
;*****

do      n0,_end_bfy

move    x:(r1),x1      y:(r6),y0
mac      x1,y0,b      x:(r6)+n6,x0      y:(r1)+,y1
macr     -x0,y1,b      a,x:(r5)+      y:(r0),a
subl     b,a          x:(r0),b      b,y:(r4)
mac      -x0,x1,b      x:(r0)+,a      a,y:(r5)
macr     -y0,y1,b      x:(r1),x1
subl     b,a          b,x:(r4)+      y:(r0),b

_end_bfy

```

```

    move    #coef,r6
    move    a,x:(r5)+n5      y:(r1)+n1,y1
    move    x:(r0)+n0,x1     y:(r4)+n4,y1

_end_grp

    andi    #fb,mr

    move    n2,b1
    lsr     b          n0,a1
    lsl     a          b1,n2
    move    a1,n0

_end_pass

endm

```

E X A M P L E

8 - 7

Execute the FFTEX1.CLD, FFTEX2.CLD, and FFTEX3.CLD programs, where each program uses a different approach to perform an 8-point FFT on a complex input sequence

In the **FFTEX1.CLD**, **FFTEX2.CLD**, and **FFTEX3.CLD** programs, the real-part elements of the input sequence are stored in X-Memory locations \$18-\$1f, and the imaginary-part elements are stored in Y-Memory locations \$18-\$1f. Each program calls the **SINCOS** macro to generate sine and cosine lookup tables, where the cosine table values are stored in X-Memory locations \$8-\$b and the sine table values are stored in Y-Memory locations \$8-\$b. Each program also calls the **BITREV** macro to store the input sequence in bit-reversed order, with the real-part bit-reverse ordered elements stored in X-Memory locations \$0-\$7 and the imaginary-part bit-reverse ordered elements stored in Y-Memory locations \$0-\$7. The FFT1, FFT2, and FFT3 macros are called by the FFTEX1.ASM, FFTEX2.ASM, and FFTEX3.ASM assembler programs, respectively, to perform an 8-point FFT on the bit-reverse ordered input sequence, where the real-part elements of the normally ordered output sequence are stored in X-Memory locations \$0-\$7 and the imaginary-part elements are stored in Y-Memory locations \$0-\$7.

In the **FFTEX1.CLD** program, the elements of the input sequence are limited to a magnitude of 0.125. The **FFTEX1.ASM** assembler program is shown in Fig. 8.17.

- (1) Load the Debug-EVM Software program.
- (2) Insert the DSP56002 Demonstration Software diskette into Drive "A."
- (3) Select the data1 window and change its radix to fractional with a mouse click on the radix button.
- (4) Select the command window and type the EVM commands:


```

load a:fftex1<return>
break #`1 p:$138<return>
go<return>
display x:$18..$1f<return>
display y:$18..$1f<return>
display x:$0..$7<return>
display y:$0..$7<return>

```
- (5) Type quit<return> if you want to exit the Debug-EVM program.

Real-part elements of the input sequence:

```
X:$0018 0.1000000 0.1000000 0.0500000 0.1000000
X:$001C 0.1000000 0.0500000 0.1000000 0.1000000
```

Imaginary-part elements of the input sequence:

```
Y:$0018 0.0000000 0.0000000 0.0000000 0.0000000
Y:$001C 0.0000000 0.0000000 0.0000000 0.0000000
```

Real-part elements of FFT output sequence:

```
X:$0000 0.7000000 0.0353554 0.0500001 -0.0353554
X:$0004 0.0000000 -0.0353554 0.0500001 0.0353554
```

Imaginary-part elements of the FFT output sequence:

```
Y:$0000 0.0000000 0.0146446 0.0500001 -0.0853555
Y:$0004 0.0000000 0.0853555 -0.0500001 -0.0146446
```

In the **FFTEX2.CLD** program, the outputs of all of the FFT butterflies are down-scaled by a factor of two. When absolute (nonscaled) results are required, a common scale-factor of 2^3 is used. The **FFTEX2.ASM** assembler program is shown in Fig. 8.18.

- (1) Load the Debug-EVM Software program.
- (2) Insert the DSP56002 Demonstration Software diskette into Drive "A."
- (3) Select the data1 window and change its radix to fractional with a mouse click on the radix button.
- (4) Select the command window and type the EVM commands:

```
load a:fftex2<return>
break #`1 p:$138<return>
go <return>
display x:$18..$1f<return>
display y:$18..$1f<return>
display x:$0..$7<return>
display y:$0..$7<return>
```

- (5) Type quit<return> if you want to exit the Debug-EVM program.

Real-part elements of the input sequence:

```
X:$0018 0.8000000 0.8000000 0.4000000 0.8000000
X:$001C 0.8000000 0.4000000 0.8000000 0.8000000
```

Imaginary-part elements of the input sequence:

```
Y:$0018 0.0000000 0.0000000 0.0000000 0.0000000
Y:$001C 0.0000000 0.0000000 0.0000000 0.0000000
```

Real-part elements of the FFT output sequence:

```
X:$0000 0.6999999 0.0353553 0.0500000 -0.0353553
X:$0004 0.0000000 -0.0353553 0.0500001 0.0353553
```

Imaginary-part elements of the FFT output sequence:

```
Y:$0000 0.0000000 0.0146446 0.0500000 -0.0853553
Y:$0004 0.0000000 0.0853553 -0.0500000 -0.0146446
```


In the **FFTEX3.CLD** program, the block floating point technique is used to accomplish conditional scaling. In this case, the scale-factor is equal to 2^k , where k is the number of passes for which down-scaling is enabled. For $N = 3$, k varies from 0 to 2. The **FFTEX3.ASM** assembler program is shown in Fig. 8.19.

Note: This example cannot be executed using the EVM. It is executed using the Motorola's SIM56000 Software Program.

- (1) Load the SIM56000 Simulator Software program.
- (2) Insert the DSP56000 Demonstration Software diskette into Drive "A."
- (3) Type the SIM56000 commands:

```
load a:fftex3<return>
break #1 pc>$218<return>
go #1<return>
display x:$18..$1f<return>
display y:$18..$1f<return>
display x:$0..$7<return>
display y:$0..$7<return>
```

- (4) Type quit<return> if you want to exit the SIM56000 program.

Real-part elements of the input sequence:

```
X:$0018 0.2000000 0.2000000 0.1000000 0.2000000
X:$001C 0.2000000 0.1000000 0.2000000 0.2000000
```

Imaginary-part elements of the input sequence:

```
Y:$0018 0.0000000 0.0000000 0.0000000 0.0000000
Y:$001C 0.0000000 0.0000000 0.0000000 0.0000000
```

Scaling factor: 2^k , where k is equal to 2.

```
r7= $0002
```

Real-part elements of the FFT output sequence:

```
X:$0000 0.3500001 0.0176777 0.0250000 -0.0176777
X:$0004 0.0000000 -0.0176777 0.0250000 0.0176777
```

Imaginary-part elements of the FFT output sequence:

```
Y:$0000 0.0000000 0.0073223 0.0250001 -0.0426776
Y:$0004 0.0000000 0.0426776 -0.0250001 -0.0073223
```

Fig. 8.17 FFTEX1.ASM Assembler Program

```
;*****
; 8-point FFT
;
;*****

include 'sincos'
include 'bitrev'
include 'fft1'
```

```

reset      equ      0
start      equ      100
points     equ      8
data       equ      0
coef       equ      8
olddata    equ      $18

;   points = number of points
;   coef   = base address of sine/cosine table
;   olddata = Starting address of input data before bit reverse
;   data   = Starting address of input data after bit reverse

olddatar    org      x:$18                ; Begin of old data
                                ; (real part)
            dc      0.1,0.1,0.05,0.1
            dc      0.1,0.05,0.1,0.1

olddataI    org      y:$18                ; Begin of old data
                                ; (imaginary part)
            dc      0,0,0,0
            dc      0,0,0,0

            sincos   points,coef

            org      p:reset
            jmp      start
            org      p:start

            bitrev   points,data,olddata
            nop

            fft1     points,data,coef,olddata
            end

```

Fig. 8.18 FFTEX2.ASM Assembler Program

```

;*****
;   8-point FFT
;
;*****

            include 'sincos'
            include 'bitrev'
            include 'fft2'

reset      equ      0
start      equ      100
points     equ      8
data       equ      0
coef       equ      8
olddata    equ      $18

;   points = number of points
;   coef   = base address of sine/cosine table
;   olddata = Starting address of input data before bit reverse
;   data   = Starting address of input data after bit reverse

```

```

      org      x:$18                ; Begin of old data
olddatar      dc      0.8,0.8,0.4,0.8      ; (real part)
      dc      0.8,0.4,0.8,0.8

      org      y:$18                ; Begin of old data
olddataI      dc      0,0,0,0              ; (imaginary part)
      dc      0,0,0,0

      sincos   points,coef

      org      p:reset
      jmp      start
      org      p:start

      bitrev   points,data,olddata
      nop

      fft2     points,data,coef,olddata
      end

```

Fig. 8.19 FFTEX3.ASM Assembler Program

```

;*****
;
;   8-point FFT
;
;*****

      include 'sincos'
      include 'bitrev'
      include 'fft3'

reset      equ      0
start      equ      100
points     equ      8
data       equ      0
coef       equ      8
olddata    equ      $18
accr       equ      $20

;   points = number of points
;   coef   = base address of sine/cosine table
;   olddata = Starting address of input data before bit reverse
;   data    = Starting address of input data after bit reverse
;   accr    = address for the ccr register

      org      x:$18                ; Begin of old data
olddatar      dc      0.2,0.2,0.1,0.2      ; (real part)
      dc      0.2,0.1,0.2,0.2

      org      y:$18                ; Begin of old data
olddataI      dc      0,0,0,0              ; (imaginary part)
      dc      0,0,0,0

```

dc	0,0,0,0
sincos	points,coef
org	p:reset
jmp	start
org	p:start
bitrev	points,data,olddata
nop	
fft3	points,data,coef,olddata,accr
end	

8.4 REFERENCES

1. Alan V. Oppenheim and Ronald W. Schaffer, *Discrete-Time Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1989.
2. Johnny R. Johnson, *Introduction to Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1989.
3. Richard A. Roberts and Clifford T. Mullis, *Digital Signal Processing*, Reading, MA: Addison-Wesley, 1987.
4. Chi-Tsong Chen, *One-Dimensional Digital Signal Processing*, New York: Marcel Dekker, 1979.
5. Roman Kuc, *Introduction to Digital Signal Processing*, New York: McGraw-Hill, 1988.
6. John G. Proakis and Dimitris G. Manolakis, *Introduction to Digital Signal Processing*, New York: Macmillan, 1988.

Designing Adaptive FIR Filters and Implementing Them on the DSP56002 Processor

In this chapter, finite impulse response adaptive filters are designed, and implemented on the DSP56002 signal processor. Adaptive filters have the ability to adjust their own parameters (coefficients) automatically. Hence, their design requires little or no a priori knowledge of the input signal or noise characteristics of the system. Adaptive filters have two inputs, $x(n)$ and $d(n)$, which are usually correlated in some manner. For the adaptive filter shown in Fig. 9.1:

1. the filter's output $y(n)$, which is computed with the current parameter estimates, is compared with the input signal $d(n)$;
2. the resulting prediction error $e(n)$ is feedback through a parameter adaptation algorithm that will produce a new estimate for the parameters; and,

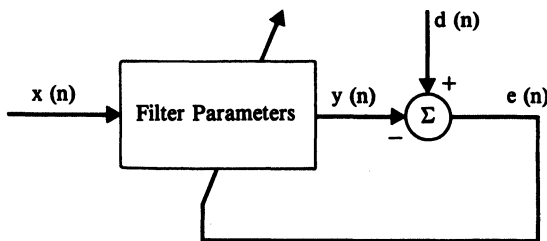


Fig. 9.1 Adaptive Filter

3. as the next input sample is received, a new prediction error will be generated, etc. The goal of the adaptive filter is the minimization of the prediction error.

Two aspects of an adaptive filter are its internal structure and its adaptation algorithm. Its internal structure can be either that of a nonrecursive (FIR) filter or that of a recursive (IIR) filter. An adaptation algorithm can be divided into two major classes: gradient algorithms and nongradient algorithms. In this chapter, a gradient algorithm is used to adjust the parameters of a FIR filter. Gradient algorithms are based on the method of steepest descent used for numerical solution of systems of linear and nonlinear equations. The Least-Mean-Square (LMS) Algorithm is probably the most widely applied gradient algorithm. The LMS algorithm adjusts the filter's parameters to minimize the mean-square error between the filter's output $y(n)$ and the desired response input $d(n)$.

This chapter begins with a review of the LMS algorithm. A DSP56002 instruction code that implements the adaptive FIR filter on the DSP Demonstration System is then described. The DSP56002 instruction code is included on the DSP56002 Demonstration Software diskette.

9.1 LMS ALGORITHM REVIEW

The signal $x(n)$ is filtered to provide an estimate of the input signal $d(n)$ according to the following equation:

$$\begin{aligned} y(n) &= \sum_{i=0}^{N-1} b_i(n)x(n-i) \\ &= B^T(n)X(n) \end{aligned} \quad (9.1)$$

where

$$B(n) = [b_0(n)b_1(n)\dots b_{N-1}(n)]^T \quad (9.2)$$

$$X(n) = [x(n)x(n-1)\dots x(n-N+1)]^T \quad (9.3)$$

The error in the estimate can be calculated as:

$$\begin{aligned} e(n) &= d(n) - y(n) \\ &= d(n) - B^T(n)X(n) \end{aligned} \quad (9.4)$$

The square of this error is equal to:

$$e^2(n) = d^2(n) - 2d(n)X^T(n)B(n) + B^T(n)X(n)X^T(n)B(n) \quad (9.5)$$

The mean-square error is the expected value of $e^2(n)$. The mean-square error can be written as:

$$E[e^2(n)] = E[d^2(n)] - 2R(x, d)B(n) + B^T(n)R(x, x)B(n) \quad (9.6)$$

where the vector of cross correlations between $X(n)$ and $d(n)$ is defined as:

$$R(x, d) = E[d(n)X(n)^T] \quad (9.7)$$

and where the correlation matrix of the input signals $X(n)$ is defined as:

$$R(x, x) = E[X(n)X(n)^T] \quad (9.8)$$

For stationary input signals, the mean-square error in Equation 9.6 is a second-order function of the parameters. The method of steepest descent seeks the minimum mean-square error by making each change in the parameter vector proportional to the gradient of the mean-square error with respect to the parameter vector. The method of steepest descent can then be described by the following equation:

$$B(n+1) = B(n) - 0.5K\nabla[E(e^2(n))] \quad (9.9)$$

where:

- $B(n+1)$ = the updated parameter vector to be used during the next sample period.
- $B(n)$ = the parameter vector during the current sample period.
- K = the loop gain that controls the rate of convergence and the filter stability. K is a positive gain.
- $\nabla[E(e^2(n))]$ = the gradient of the mean-square error with respect to the parameter vector $B(n)$.

Using Equation 9.6, the gradient of the mean-square error can be obtained as:

$$\nabla[E(e^2(n))] = -2R(x, d) + 2R(x, x)B(n) \quad (9.10)$$

By setting the gradient to zero to yield the least mean-square error, the following solution can be obtained:

$$B_{LMS} = R^{-1}(x, x)R(x, d) \quad (9.11)$$

where Equation 9.11 is usually referred to as “Wiener-Hopf” equation.

The LMS algorithm is the most popular method used to find an approximate solution to Equation 9.11, in that it does not require explicit measurements of correlation functions, nor does it require matrix inversion. The LMS algorithm can be written as:

$$B(n+1) = B(n) - 0.5K\hat{\nabla}[E(e^2(n))] \quad (9.12)$$

where $\hat{\nabla}[E(e^2(n))]$ is an estimate of the gradient of the mean-square error with respect to the parameter vector $B(n)$. This estimated gradient can be obtained by taking the gradient of a single time sample of the squared error.

$$\hat{\nabla}[E(e^2(n))] = \nabla[e^2(n)] = 2e(n)\nabla e(n) \quad (9.13)$$

From Equation 9.4,

$$\nabla e(n) = \nabla[d(n) - B^T(n)X(n)] = -X(n) \quad (9.14)$$

Thus

$$\hat{\nabla}[E(e^2(n))] = -2e(n)X(n) \quad (9.15)$$

Using Equations 9.12 and 9.15, the LMS algorithm can be written as:

$$B(n+1) = B(n) + Ke(n)X(n) \quad (9.16)$$

Equation 9.16 can be rewritten as:

$$b_i(n+1) = b_i(n) + Ke(n)x(n-i) \quad i = 0, 1, \dots, N-1 \quad (9.17)$$

where:

$b_i(n+1)$ = the value of the updated i^{th} parameter to be used during the next sample period.

$b_i(n)$ = the value of the i^{th} parameter during the current sample period.

At the beginning of the next sample period, two new input samples are shifted into the system and the process repeats. After several iterations, the FIR parameters converge to values that consistently minimize the mean square error.

The loop gain K controls the rate of convergence and stability of the FIR filter. K has an optimum value depending on the FIR tap length used. The following values, found empirically, are recommended maximum values, and we should start out with a value less than or equal to the K value shown for the corresponding FIR length chosen. The FIR tap length should be rounded to the nearest FIR tap length shown in Table 9.1.

In many digital signal processing applications such as adaptive line enhancement (ALE), the input signal $x(n)$ is a delayed version of the input signal $d(n)$. The ALE detects and tracks a moving spectral line in a broadband noise while enhancing

Table 9.1 Maximum Recommended Loop Gains

FIR Tap Length	Maximum Recommended K
≤ 32 (20 Hex)	0.75 (600000 Hex)
≤ 64 (40 Hex)	0.375 (300000 Hex)
≤ 128 (80 Hex)	0.125 (100000 Hex)
≤ 192 (C0 Hex)	0.078125 (A0000 Hex)
≤ 256 (FF Hex)	0.0703125 (90000 Hex)

the signal to noise ratios. In the following sections, it is assumed that $d(n)$ is equal to $x(n)$. It is also assumed that $d(n)$ consists of a desired signal $s(n)$ and a white noise signal $w(n)$.

$$d(n) = s(n) + w(n) \quad (9.18)$$

9.2 IMPLEMENTING AN ADAPTIVE FIR FILTER ON THE DSP56002

Equations 9.1, 9.4, and 9.17 are used to implement an adaptive FIR filter. In general, the following steps must be performed to obtain the output sample $y(n)$, the error sample $e(n)$, and the updated parameters $b_i(n+1)$:

1. Save the input sample $x(n)$ to be the first filter state.
2. Multiply the input sample $x(n)$ by $b_0(n)$ and accumulate the products of the filter states $x(n-i)$ and the coefficients $b_i(n)$ to yield the output sample $y(n)$.
3. Subtract the output sample $y(n)$ from the input sample $x(n)$ to obtain the error sample $e(n)$.
4. Update the filter parameters to obtain the next sample period parameters $b_i(n+1)$.
5. Shift the filter states to obtain the next output sample $y(n+1)$.

9.2.1 Implementing an Adaptive FIR Filter on the DSP56002 Processor

Similar to the FIR filter presented in chapter 6, steps 1–5 can be efficiently implemented on the DSP56002 processor by using its:

1. address pointers to mimic FIFO-like shifting of RAM data
2. modulo addressing capability to provide wrap-around data buffers
3. Multiply/Accumulate (MAC) instruction to both multiply two operands and add the product to a third operand in a single instruction cycle
4. data move capability in parallel with the MAC instruction to keep the multiplier running at 100% capacity
5. Repeat Next Instruction (REP) instruction to provide compact filter code

The DSP56002 processor's capability to perform modulo addressing allows an Address Register (R_n) value to be incremented (or decremented) and yet remain within an address range of size L , where L is defined by a lower and an upper address boundary.

For the adaptive FIR filter, L is equal to the number of coefficients (taps). The value $L-1$ is stored in the DSP56002 processor's Modifier Register (M_n). Because of the manner in which the DSP56002 processor's modulo addressing mechanism is implemented, the lower address boundary of L (base address) must have zeros in its J LSBs, where $2^J \geq L$; e.g., the base address value must be a multiple of 2^J . This require-

ment is applied to the DSP56002 processor's Address Register (R_n). The upper address boundary is the sum of the lower address boundary (base address) plus the modulo size minus one; e.g., the base address value plus $L-1$. Note, the upper address boundary is calculated by the DSP56002 processor and is not stored in a register.

When modulo addressing is used, the Address Register (R_n) points to a modulo (circular) data buffer located in X-Memory and/or Y-Memory. The address pointer (R_n) is not required to point at the lower address boundary; that is, it can point anywhere within the defined modulo address range L . If the address pointer increments past the upper address boundary (base address plus $L-1$ plus 1), it will wrap around to the base address.

9.2.2 DSP56002 Instructions for Implementing an Adaptive FIR Filter

The DSP56002 instructions shown in Fig. 9.2 are used to implement the adaptive FIR algorithm where:

Fig. 9.2 Assembler Code to Implement the Adaptive FIR Filter

```
;
; Adaptive FIR Filter
; x0 = Input sample
; a = Output sample
; b = Error sample * loop gain
; ntabs = number of parameter taps in the filter
; lg = loop gain
; Initialize routine
    move    #states,r1
    move    #para,r4
    move    #ntaps-1,m1
    move    m1,m4
; The following code obtains the output sample y(n):
    clr     a                x0,x:(r1)+      y:(r4)+,y0
    rep     #ntaps-1
    mac     x0,y0,a          x:(r1)+,x0      y:(r4)+,y0
    macr    x0,y0,a          x:(r1),b
; The following code obtains the product of the error sample and the loop gain:
    sub     a,b
    move    #lg,y1
    move    b,x1
    mpy     x1,y1,b
    move    b,y1
; The following code updates the parameters:
    do      #ntaps,_update
    move     y:(r4),a        x:(r1)+,x0
    mac      x0,y1,a
```

```

        move      a,y:(r4)+
_update

```

; The following instruction updates the address register R1

```

        lua      (r1)-,r1

```

1. Modulo Register M1 is programmed to the value NTAPS-1 (modulo NTAPS). Address Register R1 is programmed to point to the state variable modulo buffer located in X-Memory. Modulo Register M4 is programmed to the value NTAPS-1 (modulo NTAPS). Address Register R4 is programmed to point to the coefficient buffer located in Y-Memory. Given that the FIR filter algorithm has been executing for some time and is ready to process the input sample $x(n)$ in the Data ALU Input Register X0, the address in R4 is the base address (lower-boundary) of the coefficient buffer. The address in R1 is M, where M is greater than or equal to the lower-boundary X-Memory address and less than or equal to the upper-boundary X-Memory address. The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the DSP56002 processor's A and B Accumulators and Data ALU Input Registers X0, X1, Y0, and Y1 are as shown in Fig. 9.3.
2. The CLR instruction clears the A-Accumulator and simultaneously:
 - a. moves the input sample $x(n)$ from the Data ALU's Input Register X0 to the X-Memory location pointed to by Address Register R1, and
 - b. moves the first coefficient from the Y-Memory location pointed to by Address Register R4 to the Data ALU's Input Register Y0.

Both Address Registers R1 and R4 are automatically incremented by one at the end of the CLR instruction (postincremented).

The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A and B Accumulators and Data ALU Input Registers X0, X1, Y0, and Y1 after execution of the CLR instruction are as shown in Fig. 9.4.

3. The REP instruction regulates execution of NTAPS-1 iterations of the MAC instruction:

```

        mac      x0,y0,a      x:(r1)+,x0      y:(r4)+,y0

```

4. The MAC instruction multiplies the filter state variable in X0 by the coefficient in Y0, adds the product to the A-Accumulator, and simultaneously:
 - a. moves the next state variable from the X-Memory location pointed to by the Address Register R1 to the Input Register X0, and
 - b. moves the next coefficient from the Y-Memory location pointed to by Address Register R4 to Input Register Y0.

Both Address Registers R1 and R4 are automatically incremented by one at the end of the MAC instruction (postincremented).

The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A and B Accumulators and Data ALU Input Registers X0, X1, Y0, and Y1 after execution of the first, second, and last MAC instructions are as shown in Figs. 9.5, 9.6, and 9.7 respectively.

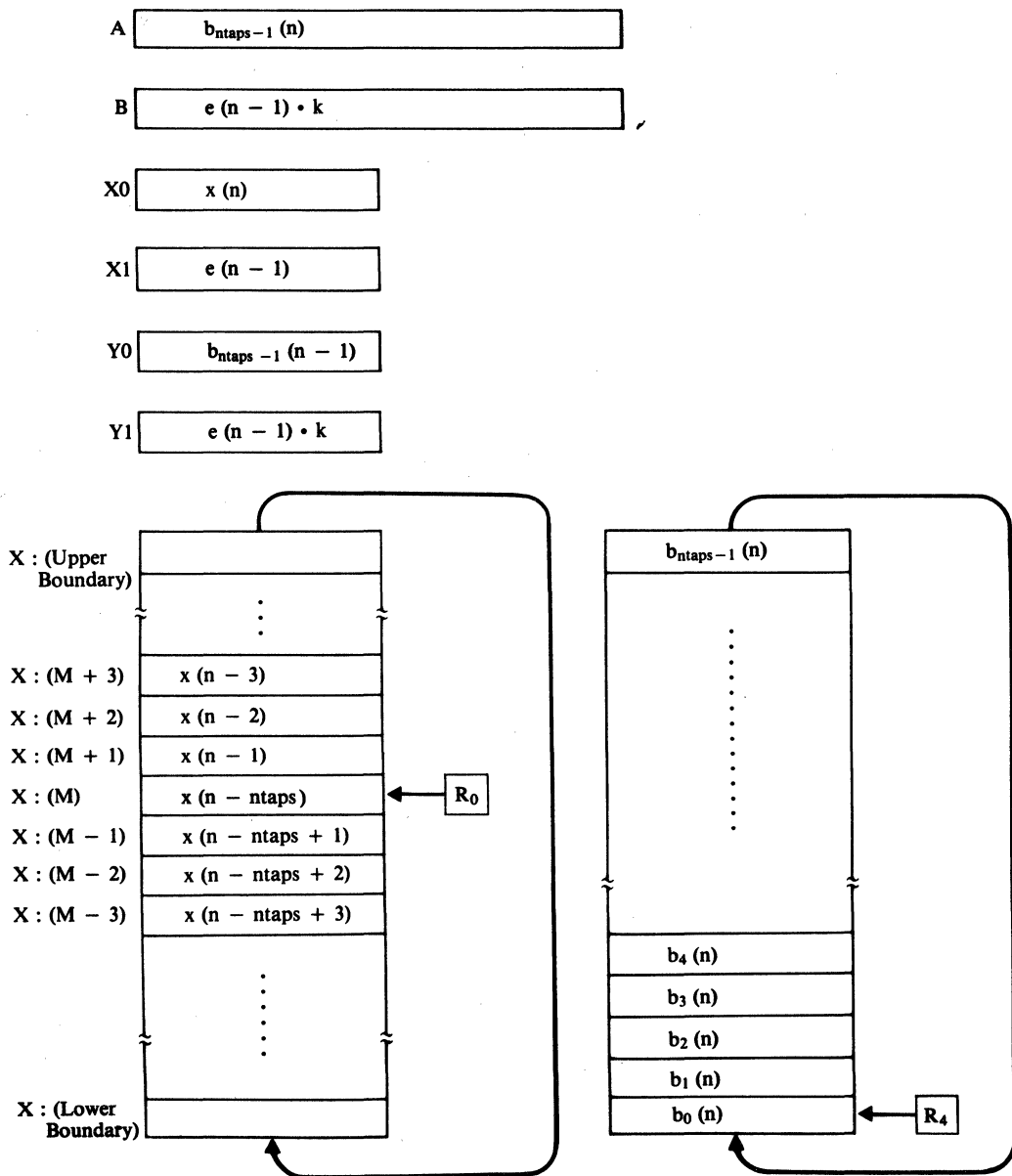


Fig. 9.3 Memory Map and Data Registers at the Beginning of the n th Iteration

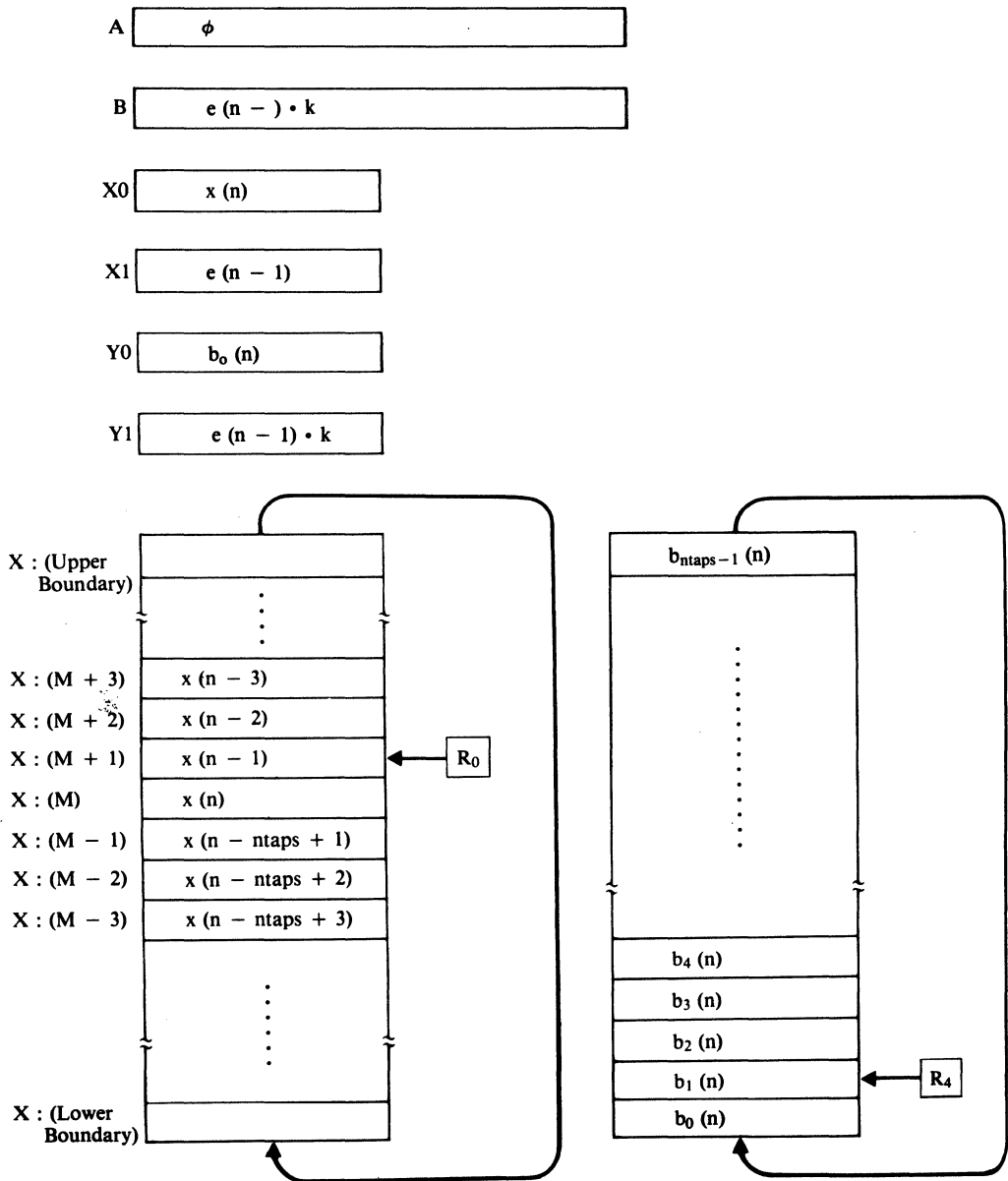


Fig. 9.4 Memory Map and Data Registers After the CLR Instruction

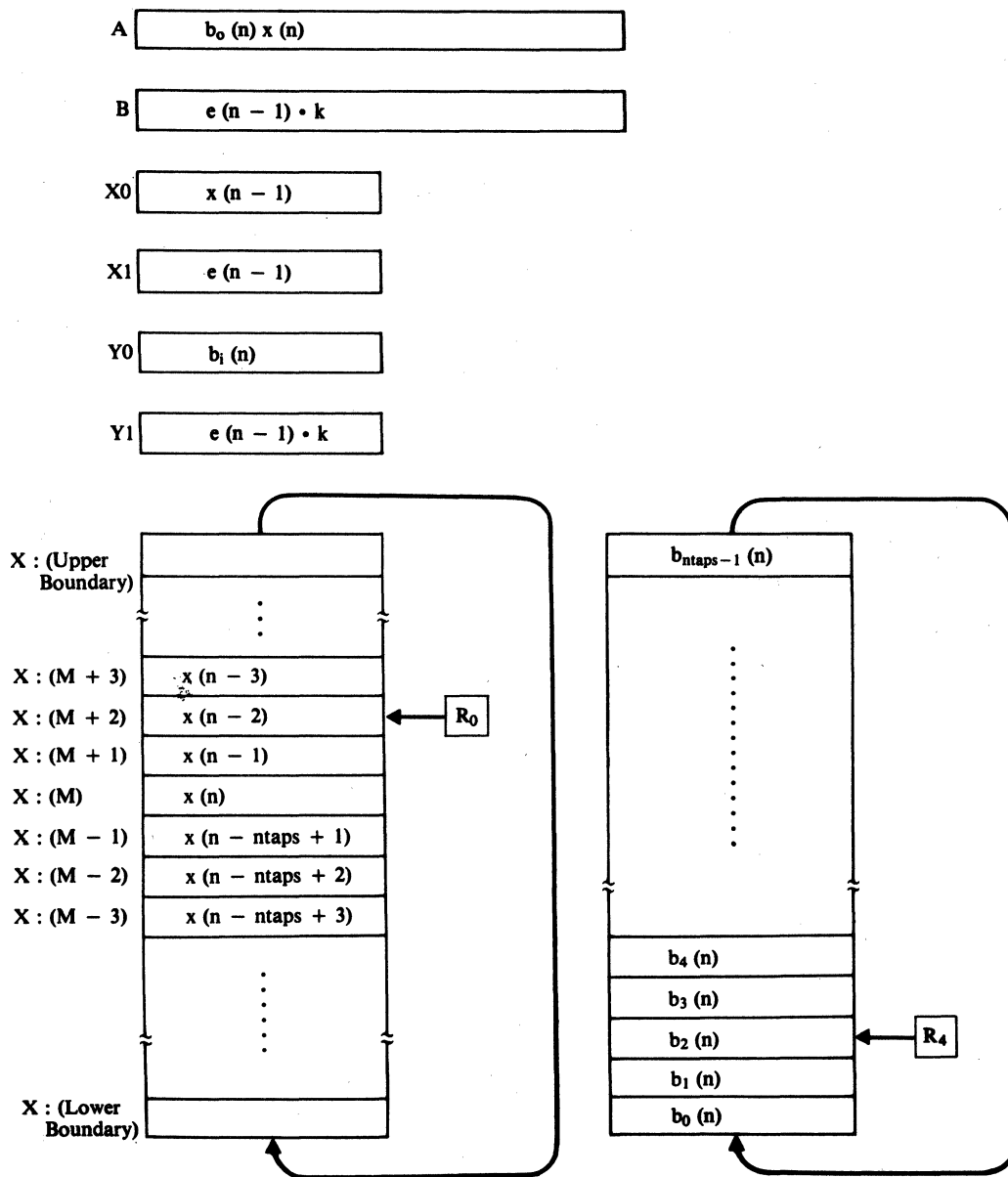


Fig. 9.5 Memory Map and Data Registers After First MAC Instruction

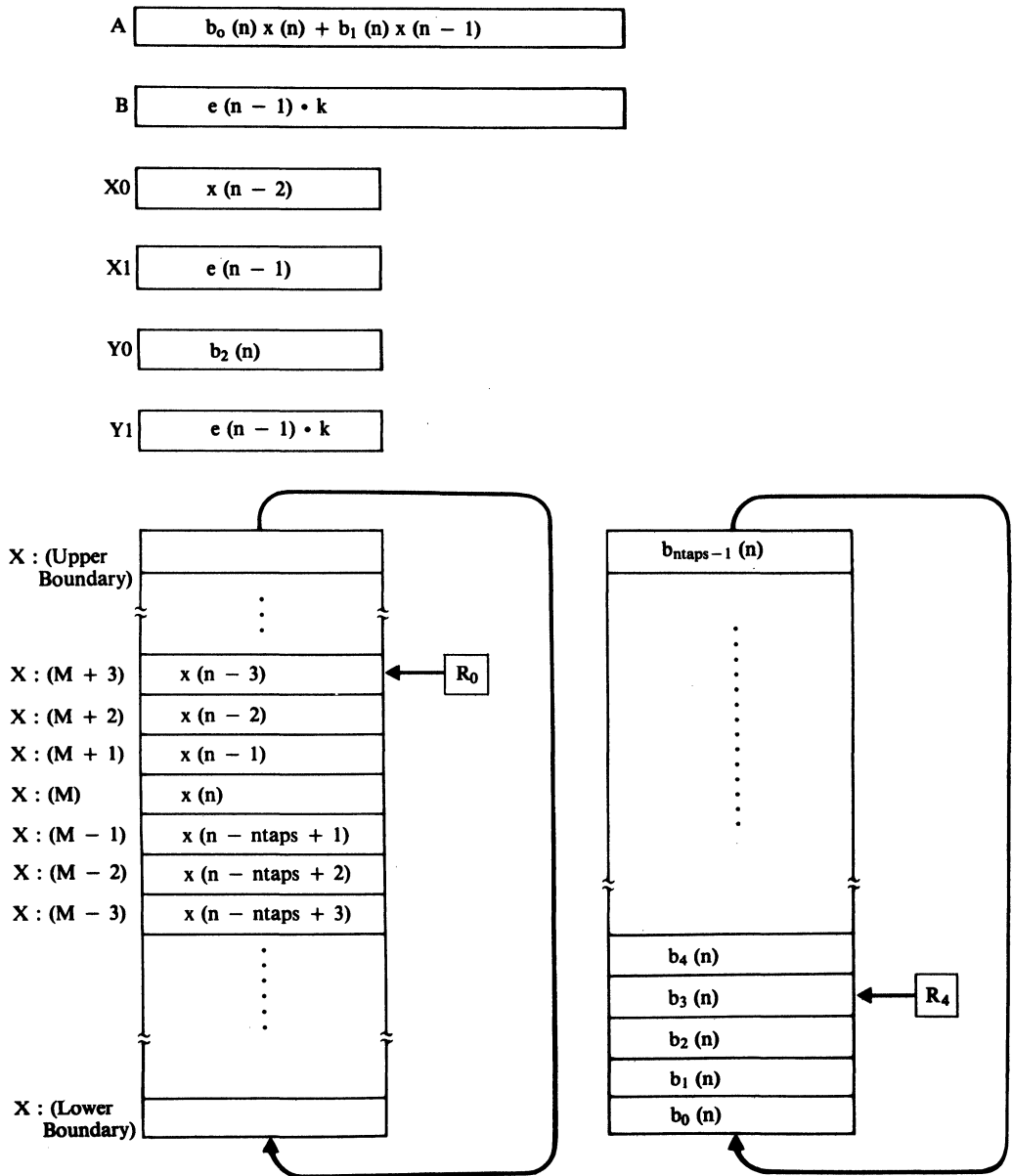


Fig. 9.6 Memory Map and Data Registers After Second MAC Instruction

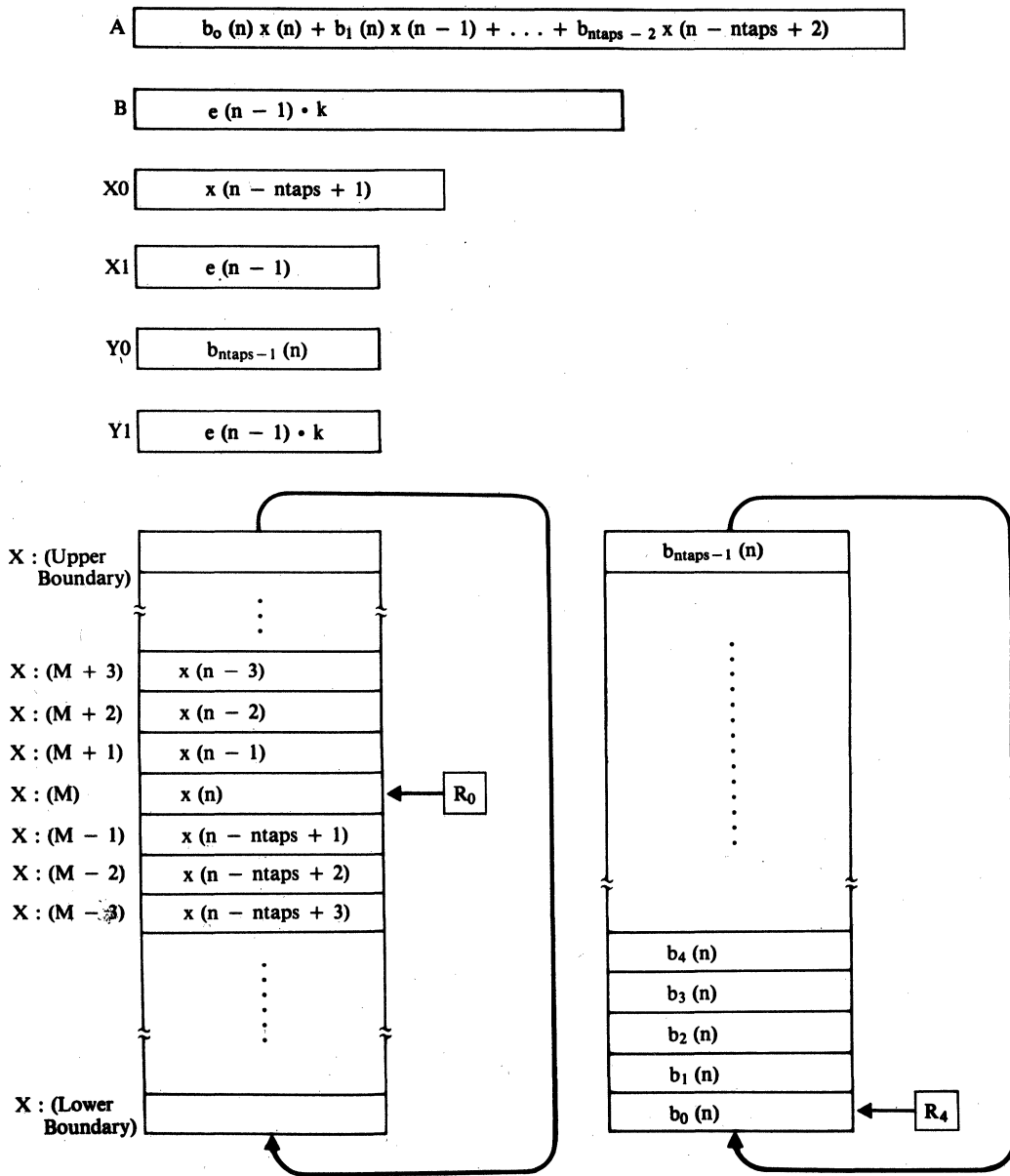


Fig. 9.7 Memory Map and Data Registers After Last MAC Instruction

Note that during the execution of the filter algorithm, Address Register R4 is postincremented a total of NTAPS times; once in conjunction with the CLR instruction and NTAPS-1 times (due to the REP instruction) in conjunction with the MAC instruction. Since the modulus for R4 is NTAPS and R4 is incremented NTAPS times, the address value in R4 wraps around and points to the coefficient buffer's lower boundary location.

Note that during the execution of the filter algorithm, Address Register R1 is postincremented a total of NTAPS times; once in conjunction with the CLR instruction and NTAPS-1 times (due to the REP instruction) in conjunction with the MAC instruction. Also note that at the beginning of the algorithm, the input sample $x(n)$ is moved from Data ALU Input Register X0 to the X-Memory location pointed to by R1. Since the modulus for R1 is NTAPS and R1 is incremented NTAPS times, the address value in R1 wraps around and points to the state variable buffer's X-Memory location M.

5. The MACR instruction calculates the final tap of the filter algorithm and performs convergent rounding of the result. The data move portion of this instruction loads the input sample $x(n)$ into the B-Accumulator. At the end of the MACR instruction, the A-Accumulator contains the filter output sample $y(n)$.

The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A and B Accumulators and Data ALU Input Registers X0, X1, Y0, and Y1 after the MACR instruction are as shown in Fig. 9.8.

6. The SUB instruction calculates the error sample $e(n)$ by subtracting the output sample $y(n)$, in the A-Accumulator, from the input sample $x(n)$, in the B-Accumulator. The error sample $e(n)$ is stored in the B-Accumulator.
7. The two MOVE instructions **move #lg,y1** and **move b,x1** transfer the loop gain K to the Data Register Y1 and the error sample $e(n)$ to the Data Input Register X1, respectively.
8. The MPY instruction multiplies the error sample $e(n)$ by the loop gain K and stores the result in the B-Accumulator. The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A and B Accumulators and Data ALU Input Registers X0, X1, Y0, and Y1 after the MPY instruction are shown in Fig. 9.9.
9. The MOVE instruction **move b,y1** transfers the product of the error sample $e(n)$ and the loop gain K to the Data Register Y1.
10. The DO instruction sets up a hardware "do loop" to update the NTAPS filter's parameters. The three instructions following the DO instruction are repeated NTAPS times.
11. The first MOVE instruction in the do loop transfers the parameter $b_i(n)$ to the A-Accumulator and the filter state $x(n-i)$ to the Data Input Register X0. Address Register R1 is incremented by one to point to the next filter state.
12. The MAC instruction multiplies the filter state, in X0, by the product of the loop gain and the error sample, in Y1, and adds the product to the A-Accumulator. The result in the A-Accumulator is the updated parameter $b_i(n+1)$.

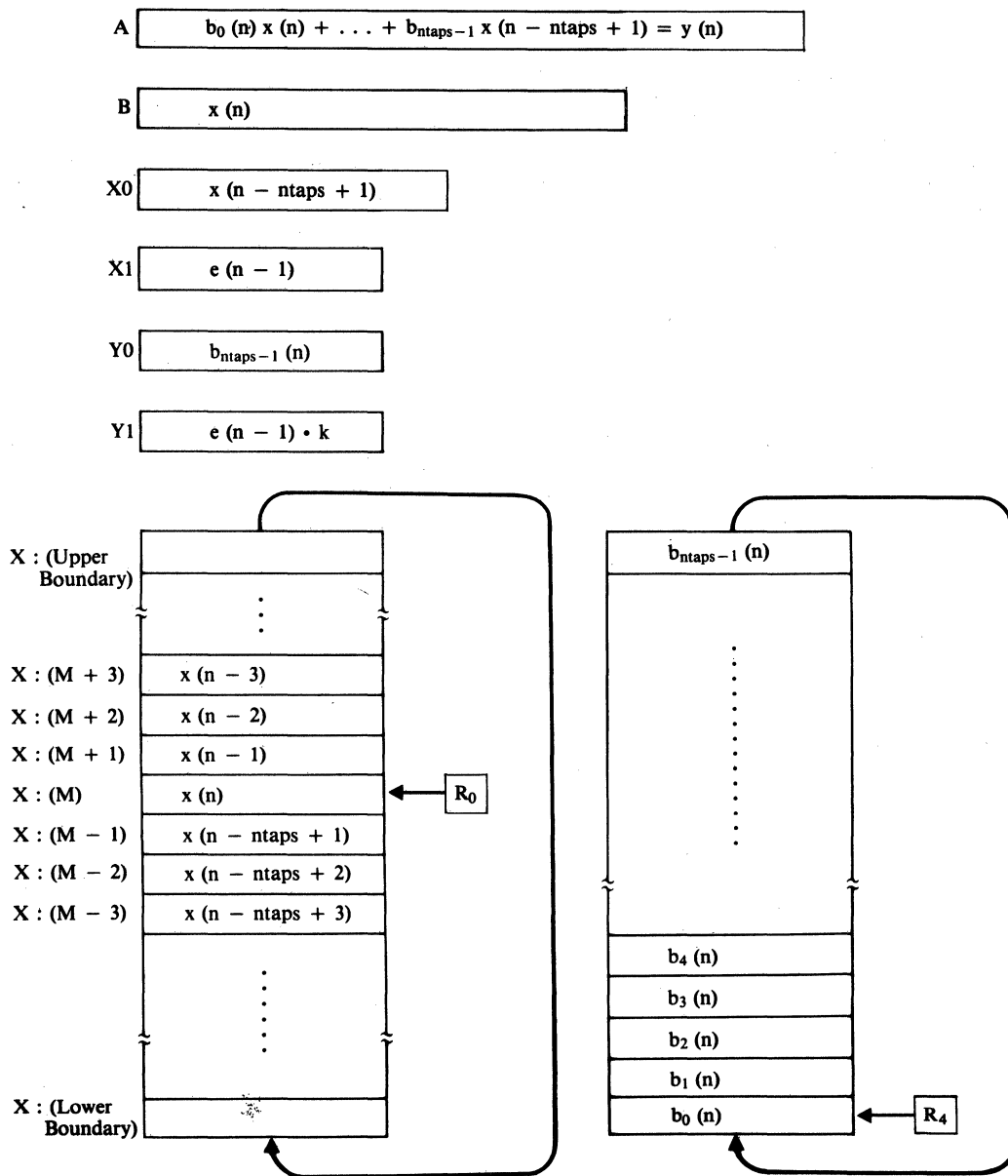


Fig. 9.8 Memory Map and Data Registers After MACR Instruction

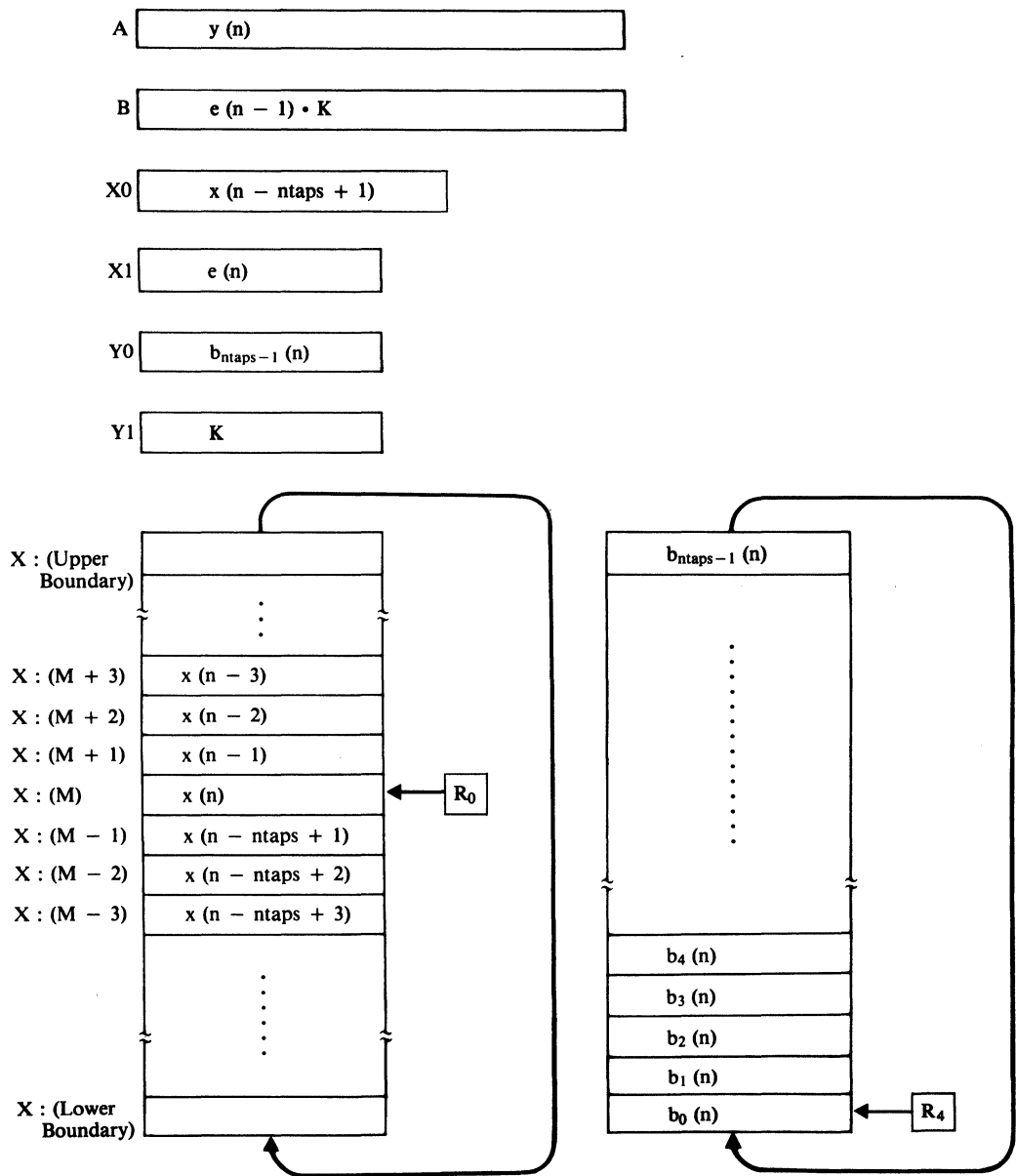


Fig. 9.9 Memory Map and Data Registers After MPY Instruction

13. The second MOVE instruction in the do loop transfers the parameter $b_i(n+1)$ to the Y-Memory location pointed to by the Address Register R4. R4 is incremented by one to point to the next filter parameter. The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A and B Accumulators and Data ALU Input Registers X0, X1, Y0, and Y1 after the first, second, and last passes of the do loop are as shown in Figs. 9.10, 9.11, and 9.12, respectively.
14. The LUA instruction decrements R1 by one. As a result, R1 then points to the state variable buffer's X-Memory location M-1. The next time the algorithm is executed, a new (next) input sample $x(n+1)$ will then overwrite the value in X-Memory location M-1. Thus FIFO-like shifting of the filter state variables is accomplished by simply adjusting the R1 address pointer. The X-Memory map for the filter states, the Y-Memory map for the coefficients, and the contents of the A and B Accumulators and Data ALU Input Registers X0, X1, Y0, and Y1 after the LUA instruction are shown in Fig. 9.13.

9.3 ADAPTIVE FIR FILTER DEMONSTRATION SYSTEM

The adaptive filtering process can be demonstrated by using the DSP system described in chapter 5 and shown in Fig. 5.1. The DSP system consists of two analog-to-digital (A/D) converters, the DSP56002 processor, and two digital-to-analog (D/A) converters. The DSP56000EVM Evaluation Module (EVM) is used to provide and control the DSP56002 processor, the two A/D converters, and the two D/A converters. The left analog input signal $x(t)$ consists of the desired input signal $s(n)$ plus a white noise signal $w(n)$. The left analog input signal $x(t)$ is first digitized using the A/D converter on the EVM. The DSP56002 processor executes the adaptive filter algorithm to process the left digitized input signal $x(n)$ and generate the left and right output signals $y_1(n)$ and $y_2(n)$. The left output signal $y_1(n)$ is the error signal. The right output signal $y_2(n)$ is the filtered version of the left digitized input signal $x(n)$, which is an estimate of the desired input signal $s(n)$. The two D/A converters on the EVM are then used to convert the left and right digital output signals $y_1(n)$ and $y_2(n)$ to the left and right analog output signals $y_1(t)$ and $y_2(t)$.

9.4 IMPLEMENTING AN ADAPTIVE FIR FILTER ON THE DEMONSTRATION SYSTEM

The STAFIR.ASM assembler program shown in Fig. 9.14 and the INIT.ASM assembler program shown in Fig. 5.15 will be used to implement the adaptive filter on the DSP system.

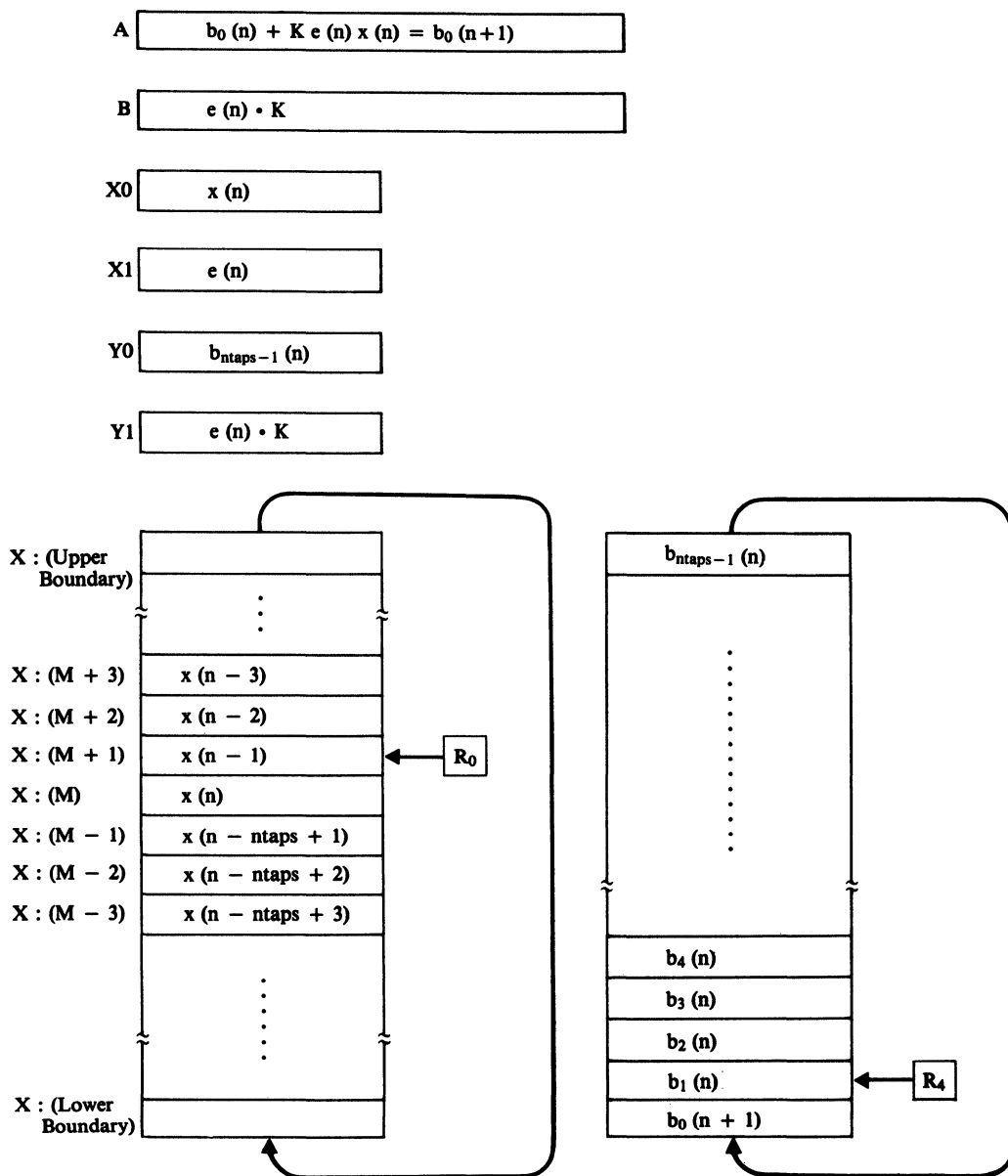


Fig. 9.10 Memory Map and Data Registers After First Pass of Do Loop

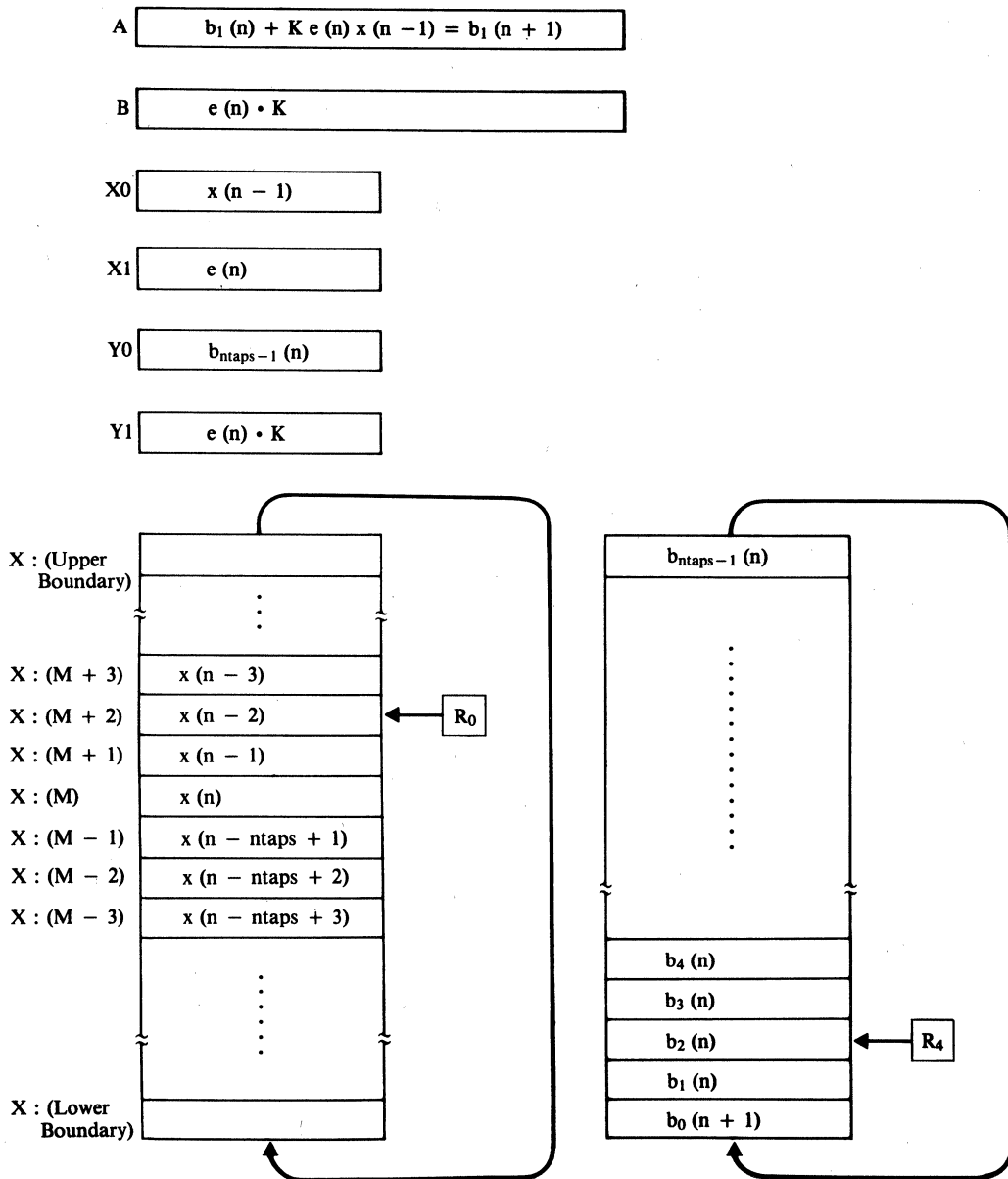


Fig. 9.11 Memory Map and Data Registers After Second Pass of Do Loop

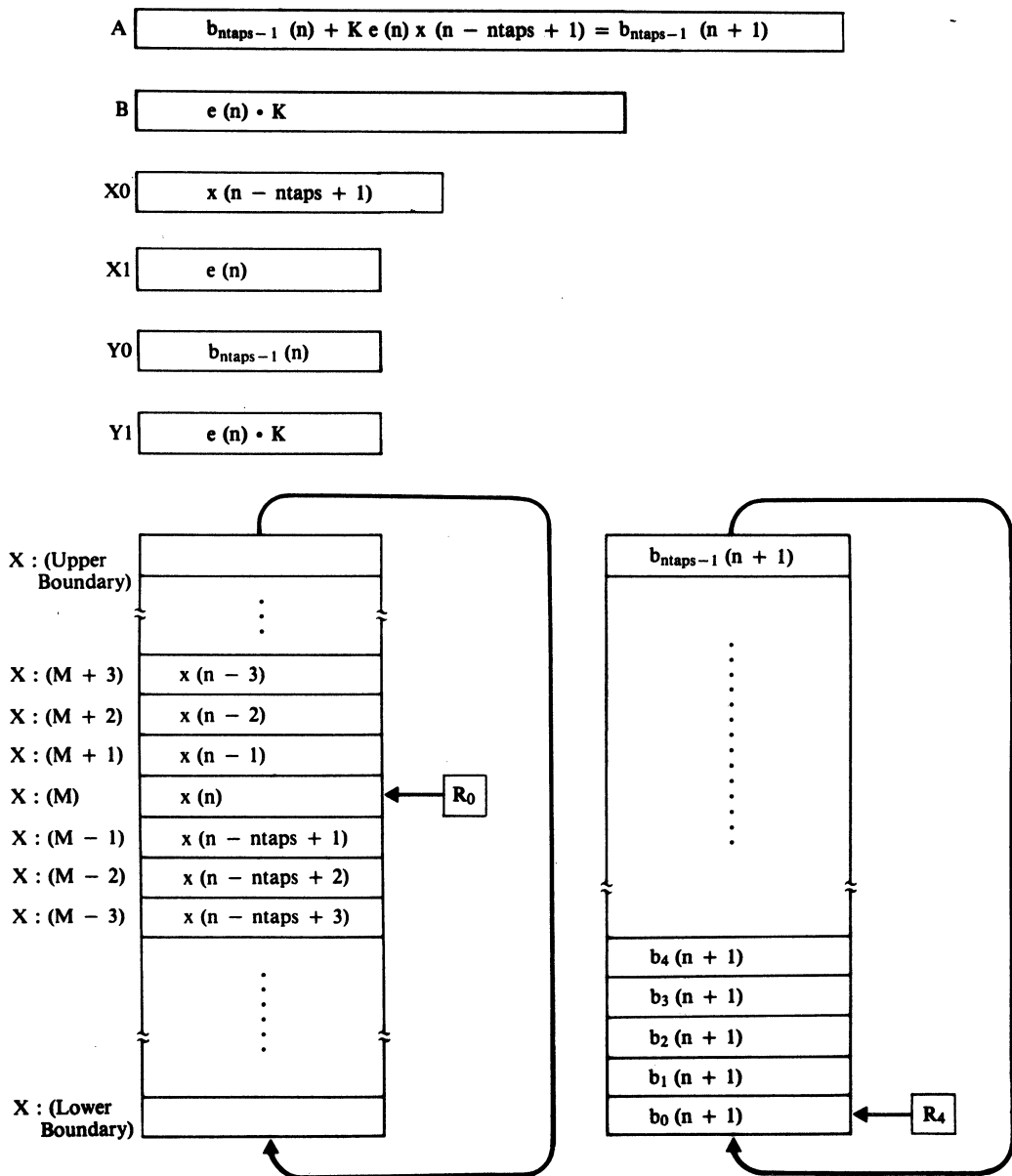


Fig. 9.12 Memory Map and Data Registers After Last Pass of Do Loop

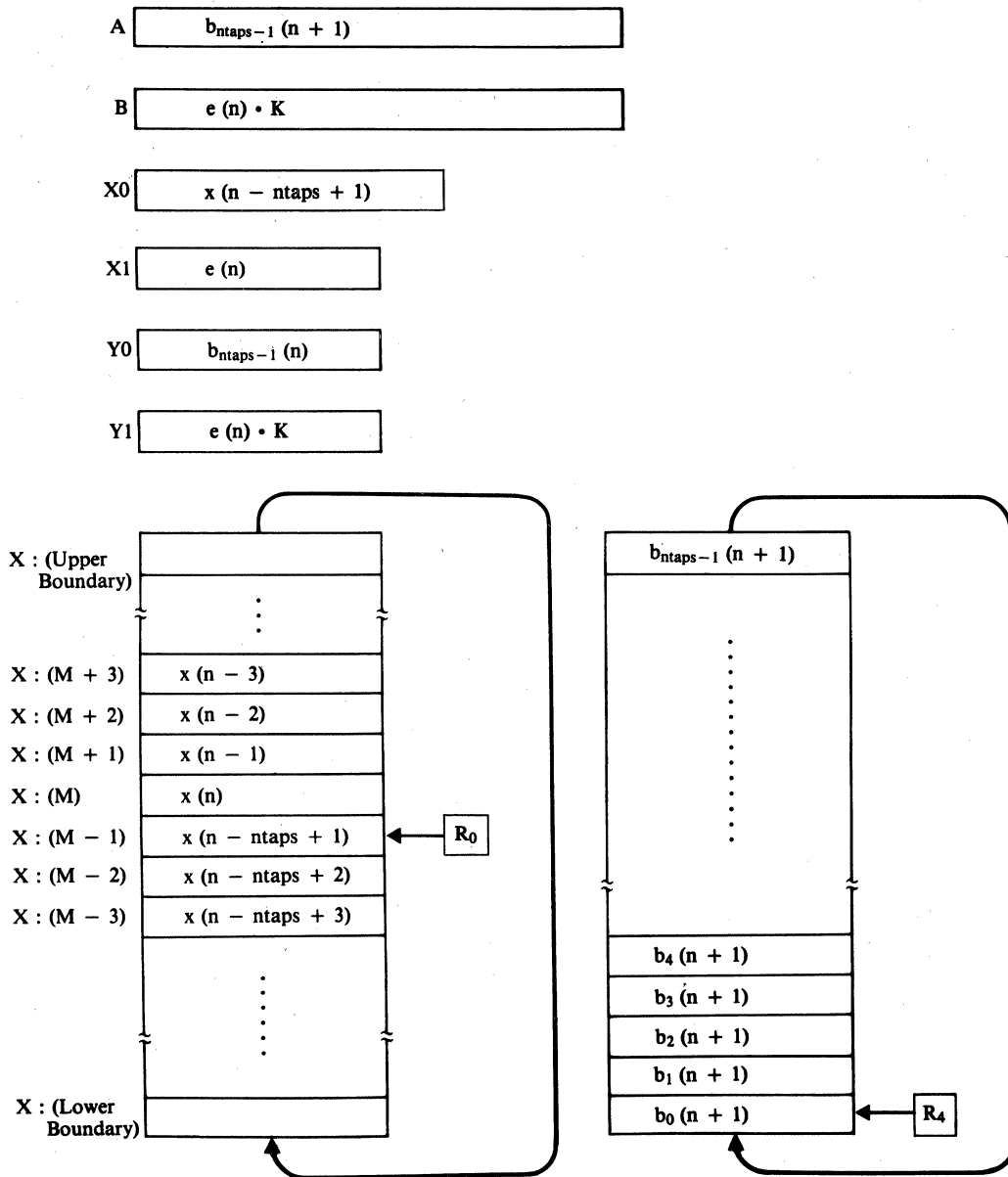


Fig. 9.13 Memory Map and Data Registers After LUA Instruction

Fig. 9.14 STAFIR.ASM Assembler Program to Implement the Adaptive FIR Filter

```

init_filter macro

    move    #states,r1    ;point to filter states
    move    #ntaps-1,m1    ;mod(ntaps)
    move    #coef,r4    ;point to filter coefficients
    move    #ntaps-1,m4    ;mod(ntaps)

    endm

stafir macro  ntaps,lg,foutput,ferror

;
;
; ntaps = number of coefficient taps in the filter
; lg = loop gain
;
;
;*****
; Right Output Channel = Output of Adaptive FIR Filter
; Left Output Channel = Error of Adaptive FIR Filter
;*****
; x0 = Input sample
; a = Output sample
; b = Error sample

; The following code obtains the output sample y(n) and
; the error sample e(n):

    clr     a             x0,x:(r1)+      y:(r4)+,y0
    rep     #ntaps-1
    mac     x0,y0,a        x:(r1)+,x0      y:(r4)+,y0
    macr    x0,y0,a        x:(r1),b

    move    a,x:foutput

    sub     a,b

    nop

    move    b,x:ferror

; The following code obtains the product of the error
; sample and the loop gain
; b = Error sample * loop gain

    move    #lg,y1
    move    b,x1
    mpy     x1,y1,b
    move    b,y1

; The following code updates the coefficients:

    do      #ntaps,_update
    move     y:(r4),a x:(r1)+,x0

```

```

    mac    x0,y1,a
    move   a,y:(r4)+
_update
; The following instruction updates the address register R1
    lua    (r1)-,r1
    nop

    move   x:foutput,b
    move   x:ferror,a

    endm

```

EXAMPLE 9 - 1

The AFIREX.ASM assembler program, shown in Fig. 9.15, implements an adaptive filter with 7 taps and 0.2 loop gain on the demonstration system. Note that this program has “include” statements that reference the STAFIR.ASM assembler program macro shown in Fig. 9.14, the INIT.ASM assembler program shown in Fig. 5.15, and the INIT1.ASM assembler program macro shown in Fig. 9.16.

Fig. 9.15 AFIREX.ASM Assembler Program to Implement the Adaptive FIR Filter

```

; right channel output = output of adaptive filter
; left channel output = error signal

    include 'init.asm'
    include 'init1.asm'
    include 'stafir.asm'

    init_system
    nop

    init_filter
    nop

data_loop

    wait_receive
    wait_word

    get_left
    get_right

    stafir ntaps,lg,foutput,ferror

    put_left
    put_right

    jmp data_loop

```

Fig. 9.16 INIT1.ASM Assembler Program

```

;*****
; Adaptive FIR Example
;*****
;

ntaps      equ      $7
lg         equ      $199999
foutput    equ      $110
ferror     equ      $111

          org      x:$100
states     dsm      ntaps      ; filter states

          org      y:$100
coef       dsm      ntaps      ; coefficients

```

A copy of the AFIREX.ASM assembler program, STAFIR.ASM assembler program, INIT.ASM assembler program, INIT1.ASM assembler program, and the AFIREX.CLD absolute object program can be found on the DSP56002 Demonstration Software diskette. A white noise generator is used to generate the white noise signal.

1. Power-down the PC, and function and white noise generators.
2. Build a summing operational amplifier to generate the input signal $x(t)$ by adding the function generator's output $s(t)$ to the white noise generator's output $w(t)$.
3. Connect the input signal $x(t)$ to the left channel of the "in" input on the EVM and to channel one of a dual trace oscilloscope.
4. Connect the two channels of the oscilloscope to "out" left and right channel outputs on the EVM.
5. Power-on and boot the PC.
6. Power-on the function and white noise generators.
7. Load the Debug-EVM software program.
8. Type `change omr 0<return>` to change the contents of the OMR register to 0.
9. From within the Debug-EVM program, load the file **afirex.cld** from the DSP56002 Demonstration Software diskette.
10. Type `go<return>` to begin execution of the program.
11. Vary the sinusoidal input frequency from 0.5 to 20 KHz and observe the left input signal, and left and right output signals on the oscilloscopes. Note the close match between the right output signal $y(t)$ and the sinusoidal input signal $s(t)$. Note that the left output signal is the error signal between the input signal $x(t)$ and output signal $y(t)$.
12. Type `force b<return>` to halt execution of the program.
13. Type `quit<return>` to exit the Debug-EVM program.

9.5 REFERENCES

1. Bernard Widrow and Samuel Stearns, *Adaptive Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1985.
2. C.F.N. Cowan and P.M. Grant, *Adaptive Filters*, Englewood Cliffs, NJ: Prentice Hall, 1985.
3. Samuel D. Stearns and Ruth David, *Signal Processing Algorithms*, Englewood Cliffs, NJ: Prentice Hall, 1988.
4. M.M. Sondhi, D.A. Berkley, "Silencing Echoes on the Telephone Network," *Proc. IEEE*, Vol. 68, No.8., Aug. 1980, pp. 948-63.

Appendices

List of Figures

Please note that the DSP56001 is obsolete. It has been replaced by the DSP56001A/56002, depending on degree of compatibility required.

A DSP56001 Interface Techniques and Examples

<i>Figure</i>	<i>Page</i>	<i>Description</i>
Fig. A.1	353	Pseudo Static RAM Auto Refresh Timing in which 2 rows are refreshed in succession. Note that either E1 or E2 can disable the device. Please refer to the data sheets specific to the PSRAMs selected for any particular application.
Fig. A.2	354	DSP56001-to-PSRAM Schematic provides two functions: it controls the refresh cycles and it generates precharge delays. This is a schematic depiction of the interface circuit.
Fig. A.3	355	PSRAM Interface State Diagram implemented in a single PAL
Fig. A.4	357	DSP56001-to-PSRAM Timing shows the operation of the controller as it progresses through a pair of successive memory accesses.
Fig. A.5	358	DSP56001-to-PSRAM PLD Definition for the ABEL™ design package which implements the state diagram in Figure A.3.
Fig. A.6	358	PSRAM Interface Initialization Code used to initialize and run a simple functionality test.
Fig. A.7	361	DRAM Memory Address Multiplexing—to reduce the package size, the row addresses and the column addresses of the DRAM cells are multiplexed onto the same pins. Latches on the device are loaded with the column and row portions of the address by the signals Column Address Strobe (CAS) and Row Address Strobe (RAS).
Fig. A.8	362	DRAM Refresh Modes are available but the $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ mode has clear advantages for DSP applications.
Fig. A.9	363	DSP56001-to-DRAM Timing shows two back to back write operations.
Fig. A.10	364	DSP56001-to-DRAM Schematic provides 256k words of expansion memory to the DSP56000 Application Development System.

<i>Figure</i>	<i>Page</i>	<i>Description</i>
Fig. A.11	365	DRAM Interface State Diagram implemented in a single PAL.
Fig. A.12	367	PLD Design File—generated by ABEL™ for the DRAM Interface
Fig. A.13	368	DRAM Interface Initialization Code provides both the initialization of the DRAM interface and a simple test of the DRAM.
Fig. A.14	370	DSP56001-to-ISA Bus Interface Schematic illustrates how simple the circuitry is to connect to the ISA bus.
Fig. A.15	371	PLD Definition for the ISA Bus Interface for the PAL22V10 shown in Figure A.14
Fig. A.16	374	DSP56001-to-ISA Bus Interface Timing shows the timing for both a read and a write operation.
Fig. A.17	376	Example Program of DSP56000 Host Interface Using C Language

B Implementing IIR/FIR Filters with Motorola's DSP56000/DSP6001

Fig. B.1	382	s-Domain Analysis of Second-Order Lowpass Analog Filter
Fig. B.2	383	Gain and Phase Response of Second-Order Lowpass Analog Filter at Various Values of Damping Factor, d
Fig. B.3	384	s-Domain Analysis of Second-Order Highpass Analog Filter
Fig. B.4	385	Gain and Phase Response of Second-Order Highpass Analog Filter at Various Values of Damping Factor, d
Fig. B.5	386	s-Domain Analysis of Second-Order Bandstop Analog Filter
Fig. B.6	387	Gain and Phase Response of Second-Order Bandstop Analog Filter at Various Values of Damping Factor, d
Fig. B.7	388	s-Domain Analysis of Second-Order Bandpass Analog Filter
Fig. B.8	389	Gain and Phase Response of Second-Order Bandpass Analog Filter at Various Values of Damping Factor, d
Fig. B.9	391	Spectrum of Bandlimited Signal Repeated at Multiples of the Sampling Frequency, f_s
Fig. B.10	395	Direct-Form Implementation of Second-Order Lowpass IIR Filter and Analytical Formulas Relating Desired Response to Filter Coefficients
Fig. B.11	396	Gain and Phase Response of Second-Order Lowpass IIR Filter
Fig. B.12	397	DSP56001 Code and Data Structures for Second-Order Direct-Form Implementation of a Lowpass IIR Filter
Fig. B.13	398	Direct-Form Implementation of Second-Order Highpass IIR Filter and Analytical Formulas Relating Desired Response to Filter Coefficients
Fig. B.14	399	Gain and Phase Response of Second-Order Highpass IIR Filter
Fig. B.15	400	DSP56001 Code and Data Structures for Second-Order Direct-Form Implementation of a Highpass IIR Filter

<i>Figure</i>	<i>Page</i>	<i>Description</i>
Fig. B.16	401	Direct-Form Implementation of Second-Order Bandstop IIR Filter and Analytical Formulas Relating Desired Response to Filter Coefficients
Fig. B.17	402	DSP56001 Code and Data Structures for Second-Order Direct-Form Implementation of a Bandstop IIR Filter
Fig. B.18	403	Gain and Phase Response of Second-Order Bandstop IIR Filter
Fig. B.19	404	Direct-Form Implementation of Second-Order Bandpass IIR Filter and Analytical Formulas Relating Desired Response to Filter Coefficients
Fig. B.20	405	DSP56001 Code and Data Structures for Second-Order Direct-Form Implementation of a Bandpass IIR Filter
Fig. B.21	406	Gain and Phase Response of Second-Order Bandpass IIR Filter
Fig. B.22	407	Summary of Digital Coefficients for the Four Basic Filter Types
Fig. B.23	408	Pole Location and Analysis of Second-Order Section
Fig. B.24	410	The Second-Order Canonic (Direct Form II) IIR Filter Network
Fig. B.25	412	Internal Node Transfer Function, $H_w(z)$, of Canonic (Direct Form II) Network
Fig. B.26	413	Internal Node Gain Analysis of Second-Order Canonic Form
Fig. B.27	415	Internal Node Gain Analysis of Second-Order Canonic Form
Fig. B.28	416	Network Diagram for Transpose-Form (Direct Form I) Implementation of a Single-Section Second-Order Filter
Fig. B.29	417	Gain Evaluation at First Internal Node, $u(n)$, of Transpose Network
Fig. B.30	418	Gain Evaluation at Second Internal Node, $v(n)$, of Transpose Network
Fig. B.31	419	Total Gain and Gain at Internal Nodes of Lowpass Transpose Filter Network Figure B.28 for $f_s = 1000\text{Hz}$
Fig. B.32	420	DSP56001 Code and Data Structures for Single-Section Second-Order Transpose Form
Fig. B.33	425	Composite Response of Cascaded Second-Order Sections for Lowpass Butterworth Filter (k^{th} Section Damping Factor According to Eqn. B.27)
Fig. B.34	426	Network Diagram for Cascaded Direct-Form Filter and Data Structures Used in Code Implementation (Three-Section Example)
Fig. B.35	427	DSP56001 Code for Cascaded Direct-Form Filter
Fig. B.36	428	Log Magnitude Plot of Example Lowpass Butterworth Filter
Fig. B.37	429	Phase Versus Frequency Plot for Example Filter
Fig. B.38	430	Zero/Pole Plot of Sixth-Order Lowpass Example Filter
Fig. B.39	431	Group Delay Versus Frequency for FDAS IIR Example

<i>Figure</i>	<i>Page</i>	<i>Description</i>
Fig. B.40	432	FDAS.OUT File of Example Filter for Cascaded Canonic Implementation
Fig. B.41	433	COEFF.OUT File of Example Filter Design-Scaled for Cascaded Canonic Implementation
Fig. B.42	434	COEFF.ASM File Generated by MGEN for Example Design and Cascaded Canonic Implementation
Fig. B.43	436	FDAS.OUT File of Example Filter for Cascaded Transpose-Form Implementation
Fig. B.44	437	COEFF.FLT File for Example Filter Design-Scaled for Cascaded Transpose Form
Fig. B.45	438	COEFF.ASM File Generated by MGEN for Example Design and Cascaded Transpose-Form Implementation
Fig. B.46	440	FIR Structure
Fig. B.47	442	Signal Data through a FIR Filter
Fig. B.48	448	Arbitrary Filter Example
Fig. B.49	449	Response Transformed from Polar Coordinates
Fig. B.50	449	FIR Coefficients from Eqn. B.36 for Filter Example
Fig. B.51	450	Roots (Zeros) of Eqn. B.57 for Filter Example
Fig. B.52	451	A 32-Point FIR Filter Example
Fig. B.53	452	Log Magnitude Response of Filter Example with Larger N Values
Fig. B.54	453	Window Function Effects on Filter Example
Fig. B.55	454	FDAS Output for FIR Bandpass Filter Example with a Kaiser Window
Fig. B.56	456	FDAS Output for FIR Bandpass Filter Example with Equiripple Design
Fig. B.57	458	FIR Filter Example

C Implementation of Fast Fourier Transforms on Motorola's Digital Signal Processors

Fig. C.1	465	Fourier Transform of a Rectangular Function
Fig. C.2	467	Windowing Effects When Windowing a Single Sine Wave
Fig. C.3	469	The FFT principle in layman's terms
Fig. C.4	470	Decimation-in-Time of an N-Point FFT
Fig. C.5	471	Decimation-in-Time FFT: Step Two
Fig. C.6	471	Decimation-in-Time FFT: Final Step (2-Point DFT)
Fig. C.7	471	An 8-point, radix-2, Decimation-in-Time FFT
Fig. C.8	472	Rearrangement of the "butterfly" building block of the DIT FFT
Fig. C.9	472	Rearrangement of the "butterfly" building block of the DIF FFT
Fig. C.10	472	Rearrangement of the DIT computation of Figure C.8
Fig. C.11	473	Decimation-in-Frequency concept
Fig. C.12	473	Complete 8-point radix-2 DIF FFT

<i>Figure</i>	<i>Page</i>	<i>Description</i>
Fig. C.13	476	Grouping of Butterflies in the FFT Calculation
Fig. C.14	478	DSP56001 Architecture Block Diagram
Fig. C.15	479	A radix-2 DIT butterfly needing less instruction cycles than a radix-2 DIF butterfly
Fig. C.16	479	The radix-2, DIT butterfly kernel on the DSP56001/2
Fig. C.17	480	A Simple, Triple-Nested DO Loop Radix-2 DIT FFT on DSP56001/2
Fig. C.18	482	DSP96002 Architectural Block Diagram. Two symmetric bus expansion ports with two channel DMA controller that blow away the speed limit on external memory access and data I/O.
Fig. C.19	483	The Radix-2, DIT FFT Butterfly Kernel on the DSP96002
Fig. C.20	484	DSP56156 architectural block diagram. Only four address registers are available. On the single data memory space, only a dual-read per one instruction is allowed.
Fig. C.21	485	The butterfly core of the DSP56156. Notice that a single write operation paralleling with an instruction always occupies a whole data move field.
Fig. C.22	488	In-place bit-reversed to normal order conversion
Fig. C.23	489	A flow diagram of two stages in a radix-2 DIT butterfly—four complex multiplications are involved in the computation.
Fig. C.24	490	A flow diagram of a Radix-4 DIT butterfly; 12 multiplications and 22 additions or subtractions are required.
Fig. C.25	491	Radix-4 DIT Butterfly takes 17 instructions on the DSP96002
Fig. C.26	497	Trivial twiddle factors in a 12-point complex radix-2 DIT FFT. The butterflies in highlighted groups can be calculated without multiplications. A, B, C, and D are radix-4 butterfly pointers.
Fig. C.27	500	Non-redundancy calculation of the Cooley-Tukey radix-2 DIT FFT with real inputs
Fig. C.28	501	Bergland algorithm has only $\log_2(N)-1$ passes and one more addition and subtraction
Fig. C.29	502	(a) Butterfly of Bergland Algorithm with $W = 1$, (b) Butterfly of Bergland Algorithm with $W \neq 1$
Fig. C.30	503	C language code that generates Bergland order tables
Fig. C.31	508	Computation of the Real-Input, DIT FFT
Fig. C.32	509	DSP56001 assembly code that calculates energy of DFT coefficients by single parameter
Fig. C.33	510	Double buffering input data so that data input can work with the FFT program concurrently

<i>Figure</i>	<i>Page</i>	<i>Description</i>
Fig. C.34	511	Block diagram of the double buffering technique. SSI/HI fast interrupt has higher priority than the MAIN or FFT program. The pointer of buffer is checked by SCI timer interrupt which has highest interrupt priority. The interval of the timer interrupt is set according to data length so that the buffer pointer can be updated accordingly.
Fig. C.35	513	The flow diagram of an 8-point discrete cosine transform. Note that the output order of the transform is scrambled.
Fig. C.36	518	Optimized Complex FFT for the DSP96002
Fig. C.37	534	Faster real FFT for the DSP96002
Fig. C.38	536	Real FFT for DSP56001/2

DSP56001 Interface Techniques and Examples

Please note that the DSP56001 is obsolete. It has been replaced by the DSP56001A/56002, depending on degree of compatibility required.

A.1 INTERFACING MOTOROLA'S DSP56001 TO PSEUDO STATIC RAM

"PSRAM combines the economies of DRAM with the straightforward interface of fully Static RAM to provide 128K bytes in a 32-pin DIP."

When the design definition of a DSP subsystem calls for a large memory space, the cost of populating this space with static RAM (SRAM) can be prohibitive. Although SRAM offers the advantages of high speed and a very simple interface, the complex structure of the SRAM storage cell results in SRAM price/density ratios which are inferior to those of dynamic RAM. Pseudo Static RAM (PSRAM) presents one possible compromise between the contradictory requirements of high density, low cost, high speed, and interface simplicity.

This section presents a simple implementation of a PSRAM interface to the DSP56001. Using an array of three 128K $\times$ 8 PSRAMs, the circuit provides access to 128K 24-bit words of data space. With the DSP56001 operating from a 33MHz clock, this memory subsystem will operate with 2 wait states for non-consecutive accesses.

A.2 PSEUDO-STATIC RAM

PSRAM combines the economies of DRAM with the straightforward interface of fully Static RAM to provide 128K bytes in a 32-pin DIP. Internally, the device contains a dynamic RAM array with on-board address multiplexing, an internal refresh row

counter, and an internal refresh timer. The memory array is divided into eight sections, each consisting of a 512 (row) $\times$ 256 (column) matrix of storage cells, forming a byte-wide memory which is 128K locations deep.

The device is pin compatible with the 128K $\times$ 8 SRAM JEDEC pinout with the exception of pin 1 (on standard SRAMs, this pin would be a no-connect; on PSRAM, it is the refresh strobe $\overline{F}$). These features enable the PSRAM to replace fully static RAM in many applications with a minimum amount of "glue."

PSRAM has two complementary enable lines, $\overline{E1}$ and $E2$. During read and write operations, these enable lines must *strobe* the address into the device. This is another difference between PSRAM and fully static RAM.

Since the PSRAM is based on DRAM storage elements, it requires a precharge delay between successive accesses and a periodic refresh. PSRAM supports three different refresh modes; CE-only refresh, auto refresh, and self refresh.

- ☛ CE-only refresh requires external hardware or software to provide periodic addressing of each of the 512 rows. Use of this method would add a considerable amount of interface hardware or would cause significant degradation to software performance.
- ☛ Self refresh can be entered after 8 ms in standby mode. In this mode the on-board refresh timer and refresh counter are used to provide the refresh sequencing. A delay slightly greater than one access cycle is required when leaving this mode before data read/write operations can proceed. This mode is useful for long standby periods, but is not suitable for device refresh during periods of normal DSP activity due to the unique timing requirements. To use this mode during idle periods would require mode selection logic as well as the circuitry associated with one of the other "active access" modes.
- ☛ Auto refresh occurs when the PSRAM is disabled by either of the device select inputs going false followed by the refresh pin $\overline{F}$ going active. For each transition of $\overline{F}$, one row of each section is refreshed and the refresh row counter is advanced in preparation for the next refresh cycle. The example presented in this note uses this mode because it requires the least amount of external logic and impacts the normal DSP software only when a data transfer contends with a refresh cycle.

Figure A.1 depicts an auto refresh cycle in which two rows are refreshed in succession. Note that either $\overline{E1}$ or $\overline{E2}$ can disable the device. Refer to the data sheets specific to the PSRAMs selected for any particular application.

A.2.1 DSP56001 Memory I/O Basics

Memory interface to the DSP56001 occurs over Port A of the processor. Port A consists of 24 bi-directional data lines (D0-D23), 16 address lines (A0-A15), three memory reference lines (PS, DS, X/Y), and two data strobes (RD, WR). Additionally, a pair of bus access control signals, Bus Request/Bus Grant (BR/BG), can be used to synchronize access requests between the processor and another device attempting to gain mastership of the bus. The bus access pins have alternate functions, Bus Strobe/Wait

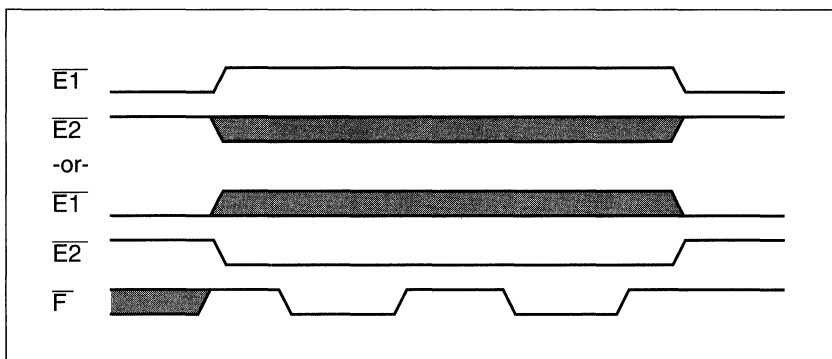


Fig. A.1 Pseudo Static RAM Auto Refresh Timing in which 2 rows are refreshed in succession. Note that either $\overline{E1}$ or $\overline{E2}$ can disable the device. Please refer to the data sheets specific to the PSRAMs selected for any particular application.

($\overline{BS}/\overline{WT}$), which allow external circuitry to insert additional wait states in external bus cycles. To minimize power consumption, the address lines remain stable until the beginning of the next external access. The memory reference signals ($\overline{PS}$, $\overline{DS}$, and $\overline{X/Y}$) are deasserted during periods when the external bus is idle, but are *not* deasserted during successive accesses to the same external memory space.

Setting bit 7 of the processor's Operating Mode Register (OMR) causes the bus access control bits to assume the Bus Strobe/Wait ($\overline{BS}/\overline{WT}$) mode. In this mode, the $\overline{BS}$ pin is asserted at the beginning of every external access and is released during T3 of each external cycle. Assertion of the $\overline{WT}$ pin during T2 *while BS is asserted* adds wait states to the bus cycle.

Wait states will continue to be inserted until two falling edges of $\overline{EXTAL}$ occur in succession with the release of $\overline{WT}$. $\overline{WT}$ should never be asserted when $\overline{BS}$ is inactive. When the DSP56001 is reading data from the bus, the data must be stable for the specified setup and hold periods before and after (respectively) the rising edge of the read strobe $\overline{RD}$. During processor write operations to the external bus, the data is valid for a specified time before and after the rising edge of the write strobe $\overline{WR}$.

These relationships are shown in the simplified PSRAM timing diagram of Figure A.4. (For DRAM timing see Figure A.9.) For more detailed information, refer to the **DSP56001 User's Manual** and the **DSP56001 Data Sheet**.

A.2.2 Memory Subsystem Overview

The circuit in Figure A.2 is designed to serve as an extension of the Motorola DSP56000 ADS Application Development Module (ADM). The Static RAM on the ADM should be configured to reside solely within the DSP56001 program space. The PSRAMs and their interface circuitry are attached to the DSP56001's Data and Address Buses via ADM connector J3.

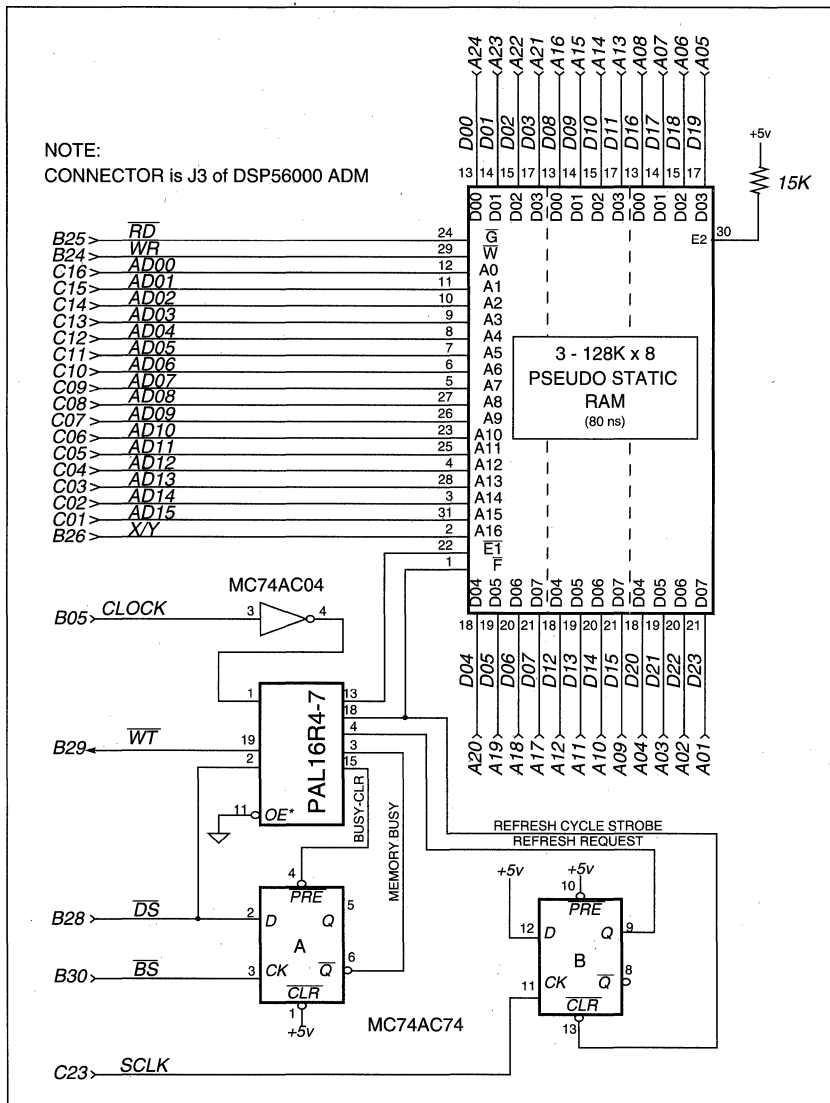


Fig. A.2 DSP56001-to-PSRAM Schematic provides two functions: it controls the refresh cycles and it generates precharge delays. This is a schematic depiction of the interface circuit.

The PSRAM bank consists of three devices. Each device provides 128K storage cells for each of 8 data bits, forming an array of 128K 24-bit words. The DSP56001 can address 64K 24-bit words in each of its two data spaces, X: memory and Y: memory. Therefore, this PSRAM array fully populates both of the processor's data spaces.

In order to minimize the component count, the refresh request timing is supplied by the processor's Serial Control Interface (SCI) clock, SCLK. Initialization software

configures this clock to provide a pulse train with a 15 μ s period. Once initialized, the generation of this signal is completely transparent to any code executing on the processor. Figure A.6 is a listing of the initialization code and a short “pass/fail” memory test routine. The value loaded into the SCI Clock Control Register (SCCR) at X:\$FFF2 will vary as a function of the system clock frequency. For a 33MHz clock, a value of \$107F will yield the desired refresh rate of 15.6 μ s per row.

A second task of the initialization software is the selection of the $\overline{BS}/\overline{WT}$ mode of operation. This mode allows an external source to insert wait states into bus cycles, and is employed by the interface when precharge and refresh delays are needed.

The interface operates from the same clock which drives the processor. In systems operating from an external clock source, this should be easy to provide. In this example, the DSP56001 clock is buffered by a CMOS inverter which subsequently drives the interface circuitry. It is essential that the device used to buffer this clock has a very high input impedance. The oscillator on the DSP56001 **cannot** drive a TTL input load.

A.2.3 Circuit Description

Figure A.2 is a schematic depiction of the interface circuit. Basically, the interface provides two functions: it controls the refresh cycles and it generates precharge delays.

Section “B” of the MC74AC74 generates a refresh request on the rising edge of SCLK and holds the request until the PAL16R4-7 controller executes a refresh cycle and resets the Flip-Flop. As shown in Figure A.3, the controller defers a refresh cycle

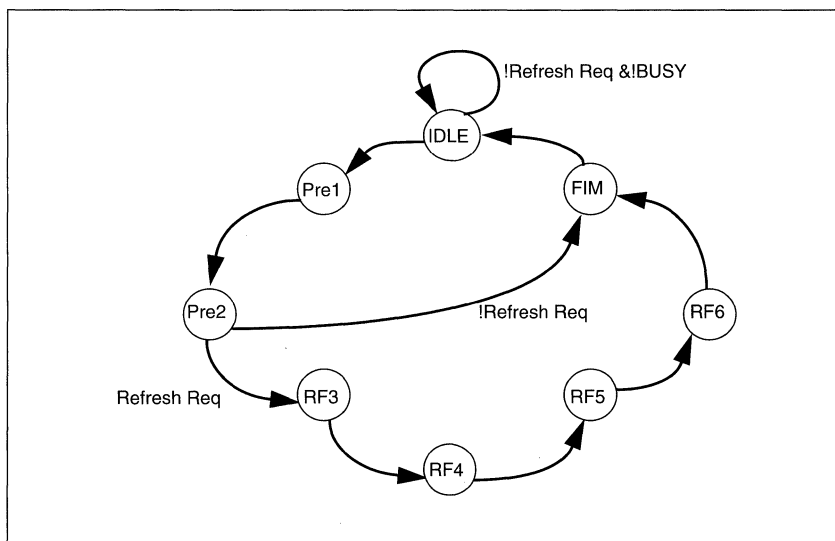


Fig. A.3 PSRAM Interface State Diagram implemented in a single PAL

until any access currently in progress completes. If the subsequent DSP56001 instruction cycle does not access this PSRAM array, this refresh is transparent. If the subsequent cycle does access this area of memory, then wait states are inserted until the refresh completes.

Section "A" of the MC74AC74 is clocked by the rising edge of $\overline{BS}$, which occurs at the end of each external bus cycle. In the event that the bus cycle which has just ended was an active cycle for the PSRAM array, the PSRAM address decode ($\overline{DS}$ in this example) will be latched into Flip-Flop "A". The PLD will receive MEMORY BUSY status, indicating that a pre-charge cycle is in progress. The PLD will hold off further PSRAM activity until sufficient precharge delay has elapsed. Note that no extra delay is seen by the DSP56001 if the subsequent cycle does not access this particular PSRAM. If multiple banks of PSRAM are used, bank interleaving strategies can result in most (or all) of the pre-charge cycles being hidden behind activity in complementary memory banks. Similarly, if the DSP56001 is executing code out of an external SRAM in another bank, the pre-charge activity would be transparent.

The ABEL^{TM1} design file for the PAL16R4-7 is a very simple Mealy type state machine (see Figure A.5). It controls the chip enabling of the PSRAM as well as the assertion of $\overline{WT}$, which goes to the DSP56001 to hold off bus activity. In addition, the machine provides resets for the external latches. The function of the PLD is shown in the state diagram of Figure A.3.

The timing diagram in Figure A.4 shows the operation of the controller as it progresses through a pair of successive memory accesses. The diagram illustrates the case where the PSRAM array was not accessed during the instruction cycle immediately preceding the start of the diagram. When operating with EXTAL at 33 MHz, the period for each T-state is 15.1 μ s. The four Tw-cycles in the first access cycle are the result of the DSP56001's Bus Control Register (BCR) being programmed to insert 2 wait states in accesses to this portion of its memory map. During the second access depicted, the controller has inserted another wait state (two Tw-periods) to allow the PSRAM to pre-charge before proceeding with the second memory cycle. Operation at 27 MHz over the full temperature range calls for 90ns access times on the PSRAMs; 70ns PSRAMs are required at 33 MHz.

Figure A.6 lists a short DSP56001 assembly-language program which initializes the memory subsystem and executes a simple test for functionality.

A.2.4 Summary

The interface in this example offers the user an option to fully populate the data storage memory of a DSP56001 system using Pseudo Static RAM instead of the SRAM which is more typical to DSP systems. The benefits of PSRAM are its favorable cost/density ratio and very simple interface requirements. This example demonstrates that the task of accommodating these interface requirements can be accomplished with very little extra logic. The benefits of PSRAM come at the expense of a slight loss of

1. ABEL is a trademark of the data I/O Corporation

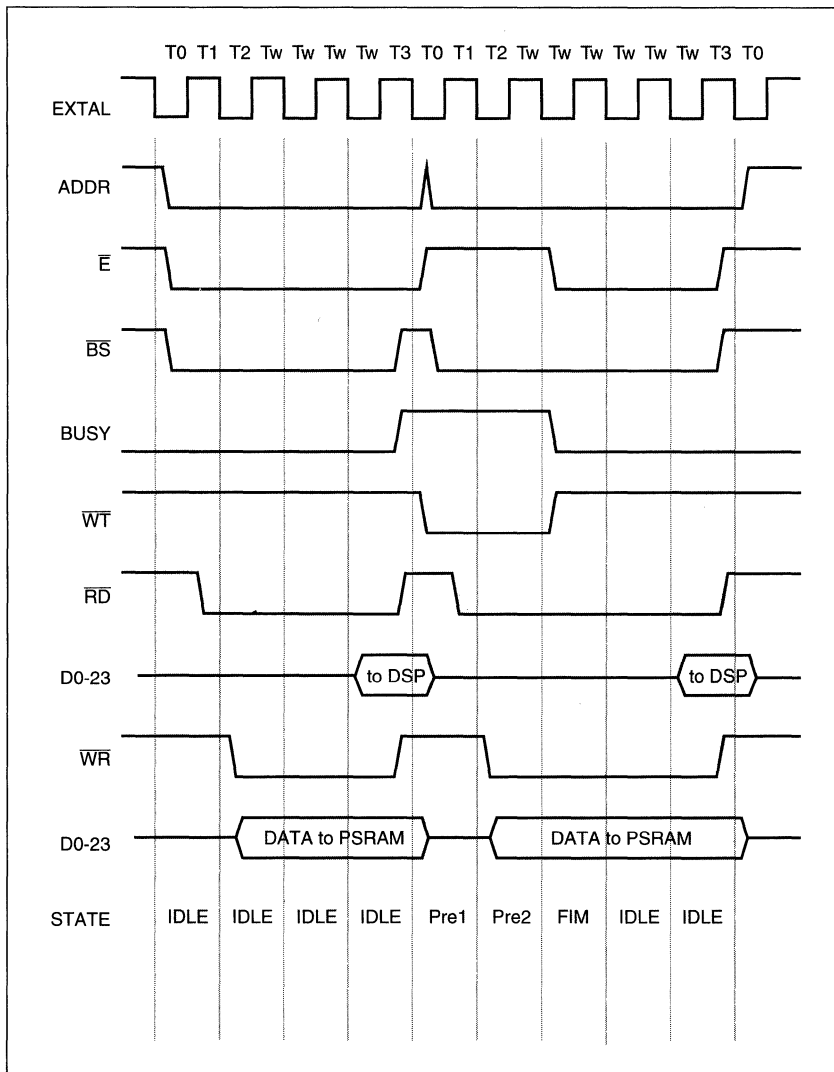


Fig. A.4 DSP56001-to-PSRAM Timing shows the operation of the controller as it progresses through a pair of successive memory accesses.

memory bandwidth. In those cases where the wait states inserted for PSRAM pre-charge and refresh are acceptable, PSRAM can reduce system cost significantly. By interleaving arrays and/or judiciously accessing the external memory, the throughput of external accesses can approach 120ns cycle times (33MHz DSP56001 and 70ns PSRAM). Operations internal to the DSP56001 and accesses to any bank(s) of fast SRAM located in P: memory will still continue at the maximum rate.

Fig. A.5 DSP56001-to-PSRAM PLD Definition for the ABEL™ design package which implements the state diagram in Figure A.3.

```

0001 module pseudo
0002 title 'Pseudo-Static RAM Timing Controller Ver.1
0003         MOTOROLA INC. 17 July 1990'
0004
0005         U01 device 'P16R4';
0006
0007 "INPUTS
0008     CLK pin 1; "DSP56001 Clock "
0009     CSin pin 2; "EXT:RAM Address decode"
0010     Busy pin 3; "BUSY F/F"
0011     Rreq pin 4; "latched request for refresh cycle"
0012
0013     OE pin 11; "OE*"
0014
0015 "OUTPUTS"
0016     Q0 pin 17; "----REGISTERED OUTPUTS----"
0017     Q1 pin 16; "State bit 0"
0018     Q2 pin 15; "State bit 1"
0019     Q3 pin 14; "State bit 2 & Busy_clr"
0020     "State bit 3 (not used)"
0021
0022     "----COMBINATORIAL OUTPUTS----"
0023     Wtn pin 19; "Bus Wait*"
0024     Fn pin 18; "Clear refresh cycle request F/F"
0025     CSout pin 13; "Chip Select for EXT:RAM"
0026
0027     High,Low = 1,0;
0028     H,L,C,K,X = 1,0,..C...K...X.;
0029
0030     Qstate = [ Q2,Q1,Q0 ];
0031     Idle = [ 1,1,1 ];
0032     Pr1 = [ 1,1,0 ];
0033     Pr2 = [ 1,0,0 ];
0034     RF3 = [ 1,0,1 ];
0035     RF4 = [ 0,0,1 ];
0036     RF5 = [ 0,0,0 ];
0037     RF6 = [ 0,1,0 ];
0038     FIM = [ 0,1,1 ];
0039
0040 state_diagram Qstate
0041     State Idle: Fn= 1; CSout = CSin; Wtn = 1;
0042                 if (!Busy & !Rreq ) THEN Idle
0043                                     ELSE Pr1;
0044     State Pr1: Fn= 1; CSout = 1 ; Wtn = CSin; goto Pr2;
0045     State Pr2: Fn= 1; CSout = 1 ; Wtn = CSin;
0046                 if (!Rreq) THEN FIM
0047                                     ELSE RF3;
0048     State RF3: Fn= 0; CSout = 1 ; Wtn = CSin; goto RF4;
0049     State RF4: Fn= 0; CSout = 1 ; Wtn = CSin; goto RF5;
0050     State RF5: Fn= 1; CSout = 1 ; Wtn = CSin; goto RF6;
0051     State RF6: Fn= 1; CSout = 1 ; Wtn = CSin; goto FIM;
0052     State FIM: Fn= 1; CSout = CSin ; Wtn = 1; goto Idle;
0053
0054     END

```

Fig. A.6 PSRAM Interface Initialization Code used to initialize and run a simple functionality test.

```

Motorola DSP56000 Macro Cross Assembler Version 3.02 90-09-06 15:06:48 psram_ex.asm Page 1
1      page 255,66,3,3,5
2      ;*****
3      ; Motorola Austin DSP Operation July 17,1990
4      ;
5      ; COPYRIGHT (C) BY MOTOROLA INC. ALL RIGHTS RESERVED

```

```

6      ;
7      ;*      ALTHOUGH THE INFORMATION CONTAINED HEREIN,      *
8      ;*      AS WELL AS ANY INFORMATION PROVIDED RELATIVE    *
9      ;*      THERETO, HAS BEEN CAREFULLY REVIEWED AND IS      *
10     ;*      BELIEVED ACCURATE, MOTOROLA ASSUMES NO           *
11     ;*      LIABILITY ARISING OUT OF ITS APPLICATION OR      *
12     ;*      USE, NEITHER DOES IT CONVEY ANY LICENSE UNDER   *
13     ;*      ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS.      *
14     ;*
15     ;
16     ;psram_ex.asm pseudo-static ram exerciser
17     ;      --- quick-and-dirty test of P-SRAM prototype board ---
18     ;
19     ;This code configures the SCI SCLK output to generate the P-SRAM
20     ;refresh timing. An incrementing pattern is written to the device
21     ;at X:$1000 and Y:$1000 and then these locations are read and compared
22     ;with the expected data. If an error is detected, an error counter
23     ;is incremented. X:0000 holds the count of errors found while accessing
24     ;X: memory and Y:0000 holds the Y:memory error count.
25     ;
26     ;This quickie only tests the interface for data transfer and refresh
27     ;interference. It does NOT exercise the refresh logic functionality.
28     ;
29     ;At the end of each pass (i.e., when the 24-bit pattern rolls over to 0)
30     ;a pass counter is incremented. This counter is at Y:0001.
31     ;
32     ;The pass counter and the error logs are located in on-chip RAM in order
33     ;to allow (limited) error analysis after any type of "crash". These
34     ;locations should be cleared before starting the test. Subsequent
35     ;restarts can continue the logging without initializing these locations.
36     ;
37     P:0100      org P:$100
38
39     P:0100 08F4BE movep #$2200,X:$FFFE ;2 wait states in X:, Y:002200
40     P:0102 08F4B0 movep #$0002,X:$FFF0 ;10-bit async mode00002
41     P:0104 08F4B2 movep #$107F,X:$FFF2 ;SCI internal CLK pinconfigured:00107F
42                                     ;TCM=RCM=0, internal clock
43                                     ;SCLK output, prescale = 1:1
44                                     ;divide fosc by 4 * (127+1)
45     P:0106 08F4A1 movep #$0004,X:$FFE1 ;SCLK/PC2 selected as SCLK0000004
46     P:0108 08F4A3 movep #$0004,X:$FFE3 ;SCLK pin configured as output 000004
47     P:010A 60F400 move  #>$1000,r0      ;r0 points to the two addresses 001000
48     P:010C 0AFA67 bset  #7,OMR          ;BS*/WT* selected
49     P:010D 221400 move  r0,r4           ;pointer reg. for Y: moves
50     P:010E 45F41B clr   b#>$000001,x1  ;constant for increment
51
52     P:0110 8A0000 loopl move a,X:(r0)a,Y:(r4);store the data in X: & Y:
53     P:0111 C08068      add  x1,b X:(r0),x0Y:(r4),y0;retrieve data and
54                                     ;...form the next data pattern
55     P:0112 200045      cmp  x0,a          ;if X: data not correct...
56     P:0113 0BF0A2      jsne X_ERR        ;...bump error count
57                                     ;
58     P:0115 200055      cmp  y0,a          ;now, check Y: data
59     P:0116 0BF0A2      jsne Y_ERR        ;...and log differences 000122
60     P:0118 21AE00      move b1,a          ;this allows data to roll-over
61     P:0119 200003      tst  a            ;check for start of new loop
62     P:011A 0BF0AA      jseq COUNT        ;and increment count if yes 000127
63     P:011C 0C0110      jmp  loopl
64
65     ;*****
66     X_ERR
67     P:011D 638000      move  X:(0),r3      ;** error handler for X:memory
68     P:011E 000000      nop                ;get last count from storage
69     P:011F 205B00      move  (r3)+         ;...can't use it yet...
70     P:0120 630000      move  r3,X:(0)      ;bump count...
71     P:0121 00000C      rts                ;save new count
72                                     ;back to the salt mine....
73
74     ;*****
75     Y_ERR
76     P:0122 6B8000      move  Y:(0),r3      ;** error handler for Y:memory

```

```

75 P:0123 000000    nop
76 P:0124 205B00    move    (r3)+
77 P:0125 6B0000    move    r3,Y:(0)
78 P:0126 00000C    rts
79
80
81 *****
82                COUNT                ;pass counter
83 P:0127 6B8100    move    Y:(1),r3
84 P:0128 000000    nop
85 P:0129 205B00    move    (r3)+
86
87 P:012A 6B0100    move    r3,Y:(1)
88 P:012B 00000C    rts
89
90                END
91
92 Errors
93 Warnings

```

A.3 A SIMPLE DYNAMIC RAM INTERFACE FOR THE DSP56001

"The high density of DRAM results from the simplicity of the storage cells; each cell consists of a single transistor and a single capacitor."

Many DSP applications, such as audio special effects, require large amounts of memory. If the system throughput can tolerate a slight reduction in memory access speed, significant cost reductions can be realized by using dynamic RAM (DRAM) in place of static RAM (SRAM). This section presents a simple implementation of a DRAM interface to the DSP56001. Using an array of six MCM514256A-P70 (256K × 4) DRAMs, the circuit provides access to 256K 24-bit words of data space. With the DSP56001 operating from a 33MHz clock, this interface can run with 2 wait states for non-consecutive accesses. For purposes of circuit simplicity, the device's fast page mode is not utilized in the following example.

A.4 DRAM BASICS

The MCM514256A DRAM is a 1 megabit part, organized as 4 sections of 256Kbits each. Each of the 4 sections is subdivided into a 512 × 512 matrix of storage cells, with each storage cell containing one bit of information. The memory cells are uniquely identified by their associated row and column numbers ("address").

In order to reduce the package size, the row addresses and the column addresses of the DRAM cells are multiplexed onto the same pins. Latches on the device are loaded with the column and row portions of the address by the signals Column Address Strobe ($\overline{\text{CAS}}$) and Row Address Strobe ($\overline{\text{RAS}}$), respectively. During a normal memory access, the cell's row number is placed on the address lines and $\overline{\text{RAS}}$ is asserted. After the specified row address hold time, the cell's column number is placed on the same address lines and $\overline{\text{CAS}}$ is asserted. This sequence is illustrated in Figure A.7.

The high density of DRAM results from the simplicity of the storage cells; each cell consists of a single transistor and a single capacitor. During write operations, the capacitor is either charged to the "one" state or discharged to the "zero" state. The charge stored by the capacitor is quite small; typical capacitor values are on the order

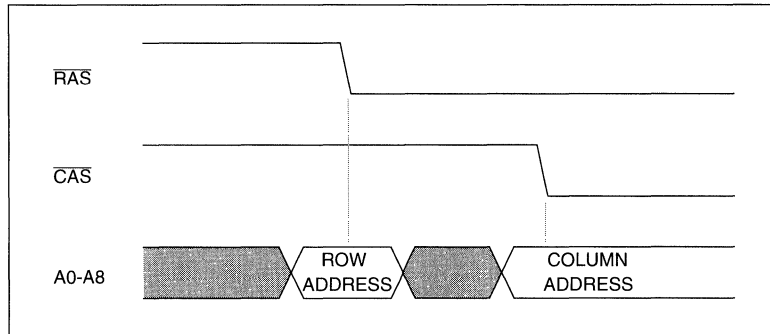


Fig. A.7 DRAM Memory Address Multiplexing—to reduce the package size, the row addresses and the column addresses of the DRAM cells are multiplexed onto the same pins. Latches on the device are loaded with the column and row portions of the address by the signals Column Address Strobe ($\overline{\text{CAS}}$) and Row Address Strobe ($\overline{\text{RAS}}$).

of 35-125 fF (fF = 1×10^{-15} farads). Due to leakage, the capacitor's charge must be periodically "refreshed" in order to retain the stored information. The DRAM circuitry will refresh all of the cells within a row whenever the row is addressed. Thus, by cycling through all 512 possible row address combinations, the entire array is refreshed. On the MCM514256A, no more than 8 ms is allowed to elapse between subsequent refreshes of any particular row. This can be accomplished by refreshing successive rows at 15.6 ms intervals ($512 \times 15.6\text{ms} = 8\text{ ms}$).

The MCM514256A supports three refresh modes: $\overline{\text{RAS}}$ only refresh, $\overline{\text{CAS}}$ -before- $\overline{\text{RAS}}$ refresh and Hidden Refresh (Figure A.8). $\overline{\text{RAS}}$ only refresh requires the processor to place successive row addresses on the address lines, which would require either more complex interface circuitry or deterministic software action (i.e., interrupts could not be allowed to delay the refresh cycle). The Hidden Refresh mode has the disadvantage of maintaining output data on the DRAM data lines, prohibiting any bus activity during the refresh cycle. $\overline{\text{CAS}}$ -before- $\overline{\text{RAS}}$ refresh utilizes an on-chip refresh row counter and three-states the device bus during the refresh cycle. The $\overline{\text{CAS}}$ -before- $\overline{\text{RAS}}$ mode is employed in this example because it requires very little external circuitry and provides for bus activity concurrent with the refresh cycle.

A requirement related to refresh is "pre-charge." During read operations, some of the charge on the cell's capacitor is lost and the memory device must re-write the information back into the cell. The DRAM automatically performs this "write back" operation after every read, but external access must be delayed until the pre-charge is complete.

Many DRAMs available today offer special access modes which can yield improved performance in specific situations. The MCM514256A DRAM supports a fast page mode in which successive accesses to cells in the same row can be read/written much faster than in normal random access situations. Although this feature would yield improved memory bandwidth in many DSP applications, the need for external address latches and comparators would add significant complexity to the circuit. Since

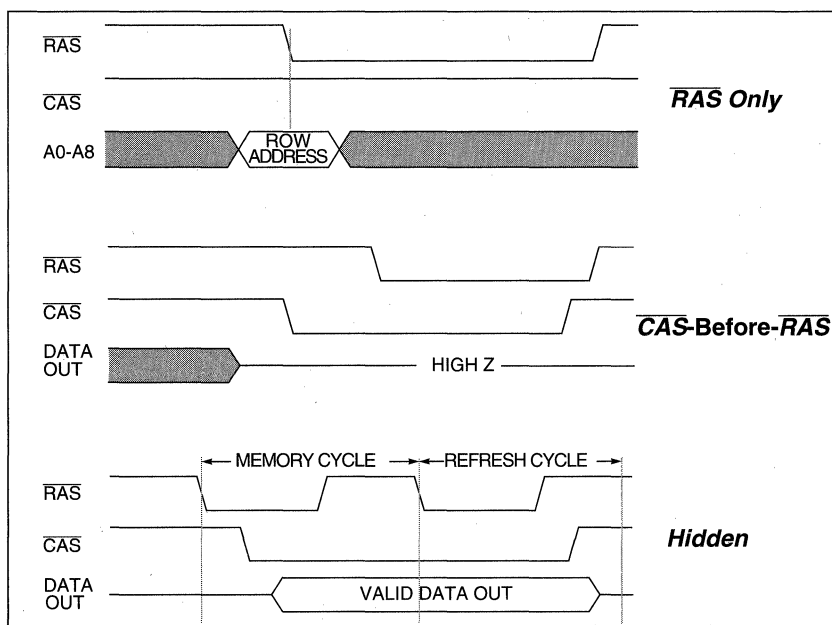


Fig. A.8 DRAM Refresh Modes are available but the $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ mode has clear advantages for DSP applications.

the design goal of this example is minimum parts count (and, therefore, minimum expense), the fast page mode of the MCM514256A is not utilized.

For more detailed information on the MCM41256A, refer to the **Motorola Memory Data Book**, DL113, Rev.5, pp.2-84 through 2-98.

NOTE: Figure A.9 shows the timing relationship between the DSP56001 Port A and a DRAM module. The Port A interface is described in **Section 1.2 DSP56001 Memory I/O Basics**.

A.4.1 Circuit Overview

The circuit in Figure A.10 is designed to serve as an extension of the MOTOROLA DSP56000ADS Application Development Module (ADM). The SRAM on the ADM should be configured to appear only in the DSP56001 P: memory space. All data memory (X: memory and Y: memory) is provided by the DRAMs on the prototype board. The DRAMs and their interface circuitry are attached to the DSP56001's Data and Address Buses via ADM connector J3.

In order to minimize the component count, the refresh request timing is supplied by the SCI clock, SCLK. Initialization software configures this clock to provide a pulse train with a 15 μs period. Once initialized, the generation of this signal is completely transparent to any code executing on the DSP56001. Figure A.13 is a listing of the initialization code and a short "pass/fail" memory test routine. The value loaded into the SCI Clock Control Register (SCCR) at X:\$FFF2 will vary as a function of the system

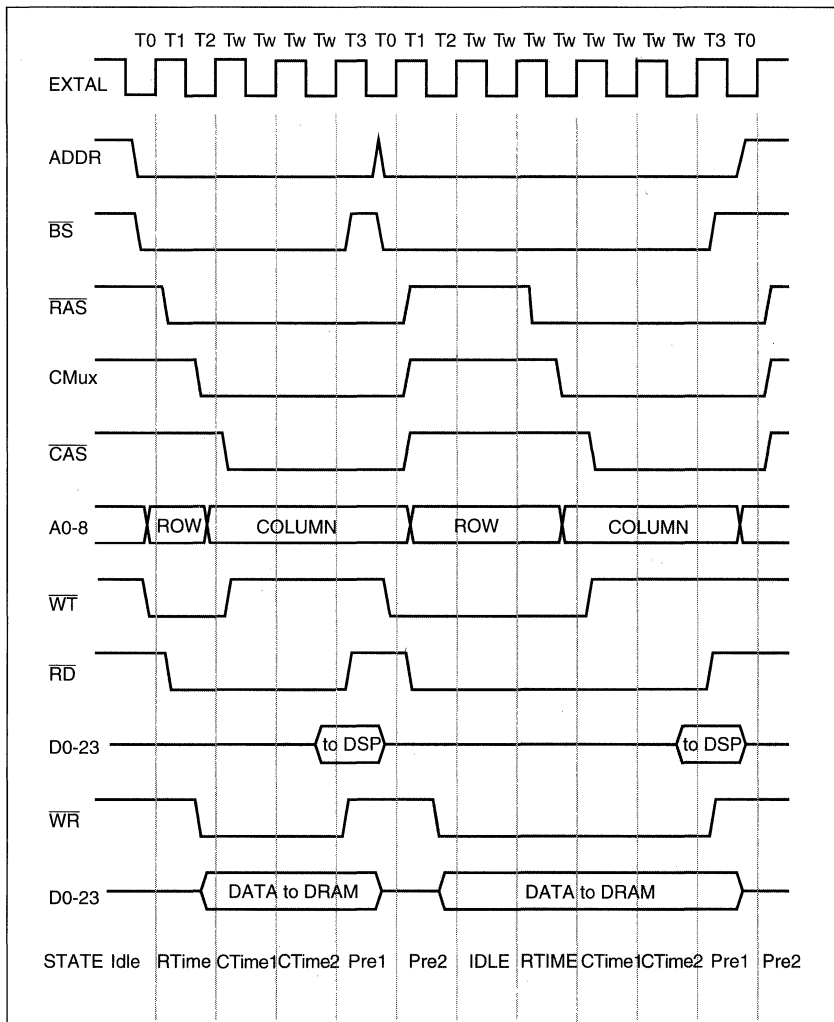


Fig. A.9 DSP56001-to-DRAM Timing shows two back to back write operations.

clock frequency. For a 33MHz clock, a value of \$107F yields the desired refresh rate of 15.6 μ s per row. A second task of the initialization software is the selection of the $\overline{\text{BS}}/\overline{\text{WT}}$ mode of operation, which allows an external source to insert wait states into bus cycles. The interface uses this feature when pre-charge and refresh delays are needed.

The memory bank consists of six MCM514256A devices. Since each device provides 256K storage cells for each of the 4 data bits, an array of 256K 24-bit words is formed. The DSP56001 can address 64K 24-bit words in each of its two data spaces, X: memory and Y: memory. This DRAM array can fully populate two of these data spaces.

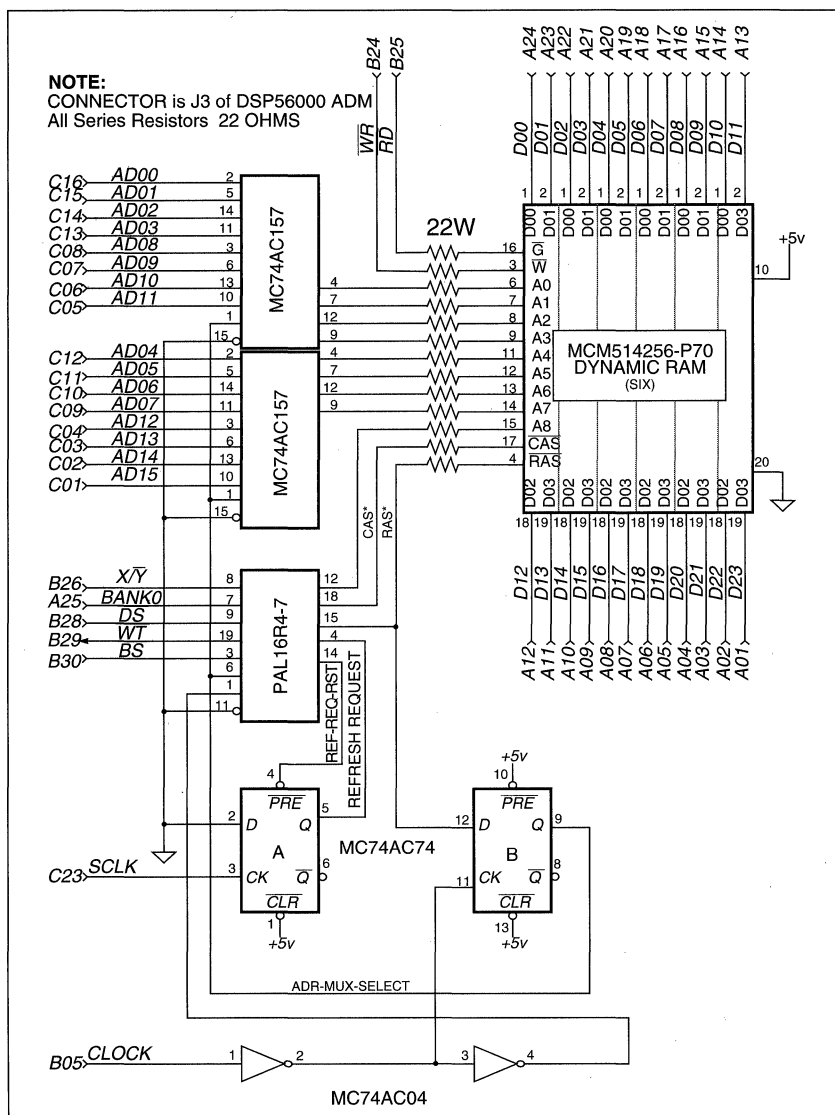


Fig. A.10 DSP56001-to-DRAM Schematic provides 256k words of expansion memory to the DSP56000 Application Development System.

To utilize this potential, a bit from the DSP56001's Port B is used as a bank selector. The configuration of this I/O bit is also handled by the initialization software.

Note of caution: accesses to the DSP56001's internal peripherals and internal data RAM do not generate external memory cycles and as such, are not subject to control by the bank selection logic.

The interface requires complementary phases of the same clock which drives the DSP56001. In systems operating from an external clock source, this should be easy to provide. In this example, the DSP56001 clock was buffered by a CMOS inverter which was subsequently used to drive the interface circuitry. It is essential that the device used to buffer this clock has a very high input impedance. The oscillator on the DSP56001 **cannot** drive a TTL input load.

Note that the Vcc and Gnd pins of the 256Kx4 DRAM do not follow the usual polarity conventions. Consult the MCM514256A data sheet for pinout information.

A.4.2 Circuit Description

The DRAM interface example provides three distinct functions:

- ☞ Memory Address Multiplexing
- ☞ Refresh Generation
- ☞ General Timing and Control

As stated earlier, DRAMs require input addresses to be subdivided into two groups — “row addresses” and “column addresses.” Referring to the schematic in Figure A.10, two MC74AC157’s multiplex 16 bits of input address onto 8 of the DRAM’s 9 address input pins. The PAL16R4-7 multiplexes the Bank Select bit and X/Y onto the 9th DRAM address input pin. Together, these 18 bits delineate two complete banks of data memory, each containing 64K 24-bit words of X: memory and 64K 24-bit words of Y: memory. In this example, bit-0 of Port B drives the bank select signal, BANK0.

Flip-Flop “A” of the MC74AC74 generates a refresh request on the rising edge of SCLK and holds the request until the PAL16R4-7 controller executes a refresh cycle and then resets the Flip-Flop. As shown in Figure A.11, the controller defers a refresh cycle until any access currently in progress completes. If the subsequent DSP56001 instruction cycle does not access this DRAM array, this refresh is transparent. If the subsequent cycle does access these DRAMs, then wait states are inserted until the refresh completes. The refresh cycle is very similar to a normal access cycle with the

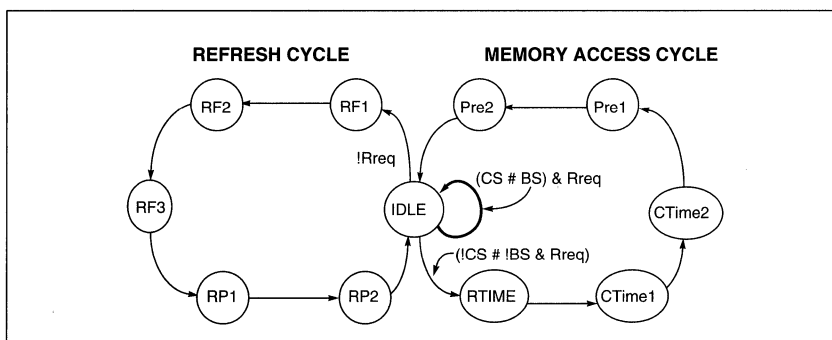


Fig. A.11 DRAM Interface State Diagram implemented in a single PAL.

exception of $\overline{\text{CAS}}$ being asserted before $\overline{\text{RAS}}$. The states of $\overline{\text{RD}}$, $\overline{\text{WR}}$ and the address lines are irrelevant.

By asserting $\overline{\text{CAS}}$ before the assertion of $\overline{\text{RAS}}$, a refresh cycle is initiated. At the completion of the refresh cycle, the refresh row counter aboard the DRAMs advances in preparation for the next refresh cycle. The interface circuit described here refreshes one row every 15 μs so that all 512 rows are refreshed within the 8 ms required by the DRAMs. In order to reduce the reflected energy on the address lines, they are terminated with 22 ohm series resistors placed as close to the drivers as is practical.

Flip-Flop "B" of the MC74AC74 is clocked from the complementary phase of EXTAL and generates the multiplexer steering control signal ADR_MUX_SELECT. This signal places the "row" portion of the address on the DRAM address lines at the beginning of a memory cycle and later selects the "column" portion of the address at the appropriate point in the cycle.

The PAL16R4-7 PLD performs the timing/control tasks required by the DRAM. The ABEL™ definition of this PLD appears in Figure A.12. The part is programmed as a Mealy-type state machine and simply advances through a sequence which selects the appropriate address portion (i.e., row or column address), generates $\overline{\text{CAS}}$ and $\overline{\text{RAS}}$ which the DRAMs require, and generates memory pre-charge delays by forcing the DSP56001 to insert wait states in any bus cycle which occurs immediately after a cycle to same DRAM array.

This pre-charge time is transparent when subsequent memory cycles do not access the same memory devices. For this reason, if more than one DRAM array is present, interleaving the arrays may yield significant improvement in the performance of the memory subsystem.

The timing diagram in Figure A.9 shows the operation of the controller as it progresses through a pair of successive memory accesses. The diagram illustrates the case where the DRAM array was not accessed during the instruction cycle immediately preceding the start of the diagram. When operating with EXTAL at 33 MHz, the length of each T-period is 15.1 μs . The four T_w -periods in the first access cycle are the result of the DSP56001's Bus Control Register (BCR) being programmed to insert 2 wait states in cycles to this portion of its memory map. During the second cycle, the controller has inserted another 2 wait states (four T_w -periods) in order to allow the DRAM to pre-charge the last row accessed before proceeding with the second memory cycle. Operation at 27 MHz over the full temperature range calls for 90ns access times on the DRAMs; 70ns DRAMs are required at 33 MHz.

Figure A.13 shows a test program that initializes the DRAM interface and then runs a simple memory test. This program provides the basic code needed by the user for using and testing the circuits in this application note.

A.4.3 Summary

The example discussed offers the user an option to add a large memory array to a DSP56001 system using DRAMs instead of the SRAM which is more typical to DSP systems. The benefit of DRAM is its favorable cost/density ratio. These benefits come at the expense of memory bandwidth. In those cases where the wait

states inserted for DRAM timing are acceptable, this example demonstrates that the task of accommodating the idiosyncracies of DRAMs can be accomplished with very little extra logic. By interleaving arrays and/or judiciously accessing the external memory, the throughput of external accesses can approach 120ns cycle times (33MHz DSP56001 and 70ns DRAM). Operations internal to the DSP56001 and accesses to any bank(s) of fast SRAM located in P: memory will still continue at the maximum rate.

Fig. A.12 PLD Design File—generated by ABEL™ for the DRAM Interface

```

0001 module   dram6
0002 title     'Dynamic RAM Timing Controller Ver.2
0003 MOTOROLA INC.   06 September 1990'
0004
0005 U01       device      'P16R4';
0006
0007 "INPUTS
0008 CLK      pin         1; "DSP56001 Clock "
0009
0010 BS       pin         3; "BUS Strobe from DSP56001"
0011 Rreq     pin         4; "latched request for refresh cycle"
0012
0013 C_Mux    pin         6;      "H = Column Mux Select"
0014 Bank0    pin         7;      "H = Select Bank 0"
0015 Xsel     pin         8;      "H = Select X:ram"
0016 CSin    pin         9;      "EXT:RAM Address decode"
0017 OE       pin        11;      "OE*"
0018
0019 "OUTPUTS"      "----REGISTERED OUTPUTS----"
0020 Q0         pin        17;      "State bit 0"
0021 Q1         pin        16;      "State bit 1"
0022 RASn       pin        15;      "State bit 2 also RASn"
0023 Rrst       pin        14;      "Refresh Request Reset"
0024           "----COMBINATORIAL OUTPUTS----"
0025 Wtn        pin        19;      "Bus Wait*"
0026 CASn       pin        18;      "Column Address Strobe for DRAM"
0027 A09        pin        12;      "DRAM address bit 9"
0028
0029 High,Low    = 1,0;
0030 H,L,C,K,X  = 1,0,..C...K...X.;
0031
0032 Qstate      = [ Rrst,RASn,Q1,Q0 ];
0033 Idle        = [ 1,1,0,0 ];
0034 Rtime       = [ 1,0,0,0 ];
0035 Ctime1      = [ 1,0,1,0 ];
0036 Ctime2      = [ 1,0,1,1 ];
0037 Pre1        = [ 1,1,1,1 ];
0038 Pre2        = [ 1,1,1,0 ];
0039
0040 RF1         = [ 0,1,0,0 ];
0041 RF2         = [ 0,0,0,0 ];
0042 RF3         = [ 0,0,1,0 ];
0043 RF4         = [ 0,0,1,1 ];
0044 RF5         = [ 0,0,0,1 ];
0045 RP1         = [ 0,1,0,1 ];
0046 RP2         = [ 1,1,0,1 ];
0047
0048 XX1         = [ 0,1,1,0 ]; "just in case it wakes up lost..."
0049 XX2         = [ 0,1,1,1 ];
0050 XX3         = [ 1,0,0,1 ];
0051
0052
0053 state_diagram Qstate
0054 State Idle:  CASn = 1;Wtn = !(CSin & !BS);
0055              if ((CSin # BS) & Rreq) THEN Idle
0056              if (!CSin & !BS & Rreq) THEN Rtime

```

```

0057      if ( !Rreq)                THEN RF1;
0058 State Rtime:  CASn = 1;Wtn = !(ICSin & !BS);goto Ctime1;
0059 State Ctime1: CASn = 0;Wtn = 1;      goto Ctime2;
0060 State Ctime2: CASn = 0;Wtn = 1;      goto Prel;
0061 State Prel:   CASn = 1;Wtn = !(ICSin & !BS);goto Pre2;
0062 State Pre2:   CASn = 1;Wtn = !(ICSin & !BS);goto Idle;
0063
0064      " --- Refresh States --- "
0065 State RF1:     CASn = 0; Wtn = !(ICSin & !BS); goto RF2;
0066 State RF2:     CASn = 0; Wtn = !(ICSin & !BS); goto RF3;
0067 State RF3:     CASn = 0; Wtn = !(ICSin & !BS); goto RF4;
0068 State RF4:     CASn = 1; Wtn = !(ICSin & !BS); goto RF5;
0069 State RF5:     CASn = 1; Wtn = !(ICSin & !BS); goto RP1;
0070 State RP1:     CASn = 1; Wtn = !(ICSin & !BS); goto RP2;
0071 State RP2:     CASn = 1; Wtn = !(ICSin & !BS); goto Idle;
0072
0073 State XX1:     goto Idle;          "if lost, go home PAL..."
0074 State XX2:     goto Idle;
0075 State XX3:     goto Idle;
0076
0077 equations
0078     A09 = (Bank0 & C_Mux) # (Xsel & !C_Mux);
0079
0080 END

```

Fig. A.13 DRAM Interface Initialization Code provides both the initialization of the DRAM interface and a simple test of the DRAM.

Motorola DSP56000 Macro Cross Assembler Version 3.02 90-09-06 10:54:50 dram_ex.asm

```

1      page      255,66,3,3,5
2      ;*****
3      ;      Motorola Austin DSP Operation   August 15,1990
4      ;
5      ;      COPYRIGHT (C) BY MOTOROLA INC, ALL RIGHTS RESERVED
6      ;
7      ;      ALTHOUGH THE INFORMATION CONTAINED HEREIN,      *
8      ;      AS WELL AS ANY INFORMATION PROVIDED RELATIVE    *
9      ;      THERETO, HAS BEEN CAREFULLY REVIEWED AND IS     *
10     ;      BELIEVED ACCURATE, MOTOROLA ASSUMES NO          *
11     ;      LIABILITY ARISING OUT OF ITS APPLICATION OR      *
12     ;      USE, NEITHER DOES IT CONVEY ANY LICENSE UNDER   *
13     ;      ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS.      *
14     ;
15     ;
16     ;      dram_ex.asm dynamic ram exerciser
17     ;      ---- quick-and-dirty test of DRAM prototype board ----
18     ;
19     ;This code configures the SCI SCLK output to generate the P-SRAM
20     ;refresh timing. An incrementing pattern is written to the device
21     ;at X:$1000 and Y:$1000 and then these locations are read and
22     ;compared with the expected data. If an error is detected, an
23     ;error counter is incremented. X:0000 holds the count of errors
24     ;found while accessing X: memory and Y:0000 holds the Y:memory
25     ;error count.
26     ;This quickie only tests the interface for data transfer and
27     ;refresh interference. It does NOT exercise the refresh logic
28     ;functionality. Bit 0 of PORT B is used to select between two
29     ;banks of 64k x 24 X 2,but this is not used in this exercise.
30     ;At the end of each pass (i.e., when the 24-bit pattern rolls over
31     ;to 0) a pass counter is incremented. This counter is at Y:0001.
32     ;The pass counter and the error logs are located in on-chip RAM
33     ;in order to allow (limited) error analysis after any type of
34     ;"crash". These locations should be cleared before starting the
35     ;test. Subsequent restarts can continue the logging without
36     ;initializing these locations.
37     ;
38     P:0100    org P:$100
39     P:0100 08F4BE movep #$2200,X:$FFFE ;2 wait states in X:, Y:002200

```

```

40 P:0102 08F4A2 movep #1,X:$FFE2      ;Port B, Bit 0 is output 000001
41 P:0104 08F4A4 movep #0,X:$FFE4      ;Port B data is all 0's 000000
42 P:0106 08F4A0 movep #0,X:$FFE0      ;Port is G.P I/O 000000
43 P:0108 08F4B0 movep #$0002,X:$FFF    ;10-bit async mode 000002
44 P:010A 08F4B2 movep #$107F,X:$FFF2;SCI internal CLK pin configured 00107F
45                                     ;TCM=RCM=0, internal clock
46                                     ;SCLK output, prescale = 1:1
47                                     ;divide fosc by 4x(127+1)
48 P:010C 08F4A1 movep #$0004,X:$FFE1;SCLK/PC2 selected as SCLK000004
49 P:010E 60F400 move #>$1000,r0        ;r0 points to the two addresses 001000
51 P:0110 0AFA67 bset #7,OMR            ;BS*/WT* selected
52 P:0111 221400 mover 0,r4             ;pointer reg. for Y: moves
53 P:0112 45F41B clrb #>$000001,x1      ;constant for increment000001
54
55 P:0114 8A0000 loop1 move a,X:(r0)a,Y:(r4) ;store the data in X: & Y:
56 P:0115 C08068 add x1,bX:(r0),x0Y:(r4),y0 ;retrieve data and
57                                     ;...form the next data pattern
58 P:0116 200045 cmp x0,a                ;if X: data not correct...
59 P:0117 0BF0A2 jsne X_ERR              ;...bump error count000121
60 P:0119 200055 cmp y0,a                ;now, check Y: data
61 P:011A 0BF0A2 jsne Y_ERR              ;...and log differences000126
62 P:011C 21AE00 move b1,a               ;this allows data to roll-over
63 P:011D 200003 tst a                   ;check for start of new loop
64 P:011E 0BF0AA jseq COUNT              ;...and increment count if yes00012B
65 P:0120 0C0114 jmp loop1
66
67                                     ;*****
68 X_ERR                                ;** error handler for X:memory **
69 P:0121 638000 move X:(0),r3            ;get last count from storage
70 P:0122 000000 nop                     ;...can't use it yet...
71 P:0123 205B00 move (r3)+               ;bump count...
72 P:0124 630000 move r3,X:(0)           ;save new count
73 P:0125 00000C rts                      ;back to the salt mine...
74
75                                     ;*****
76 Y_ERR                                ;** error handler for Y:memory **
77 P:0126 6B8000 move Y:(0),r3
78 P:0127 000000 nop
79 P:0128 205B00 move (r3)+               ; refer to X-ERR for comments
80 P:0129 6B0000 move r3,Y:(0)
81 P:012A 00000C rts
82
83                                     ;*****
84 COUNT                                ;pass counter
85 P:012B 6B8100 move Y:(1),r3
86 P:012C 000000 nop
87 P:012D 205B00 move (r3)+               ; refer to X-ERR for comments
88 P:012E 6B0100 move r3,Y:(1)
89 P:012F 00000C rts
90
91 END

```

A.5 A SIMPLE ISA BUS INTERFACE FOR THE DSP56001

"The 8 registers which comprise the DSP56001 Host Interface are mapped into the ISA bus I/O space. . .

Communications with the DSP56001, including program bootstrapping, are accomplished via I/O reads and I/O writes to the appropriate register."

The Host Port of the DSP56001 provides much of the logic necessary for interfacing this device to another processor. With very little external logic, this port can be used to interconnect the DSP56001 and an ISA Bus host processor (i.e., a PC-Clone). This brief note describes one implementation of such an interface using only two external parts.

A.5.1 Interface Circuit Overview

The interface consists of a single PAL22V10 and one MC74ACT245 octal data transceiver. The PLD generates the control signals required by the Host Interface of the DSP56001 (H $\overline{EN}$, HR/W) as well as the boot mode selection during reset. The schematic of the interface appears in Figure A.14. The PLD definition is shown in Figure A.15.

The MC74ACT245 buffers the data lines between the Host Interface and the ISA Bus. The Host Interface address lines are not buffered in this example because the DSP56001 load to these lines is equivalent to that of a typical CMOS buffer. In some cases, adding a buffer to these lines might be desirable.

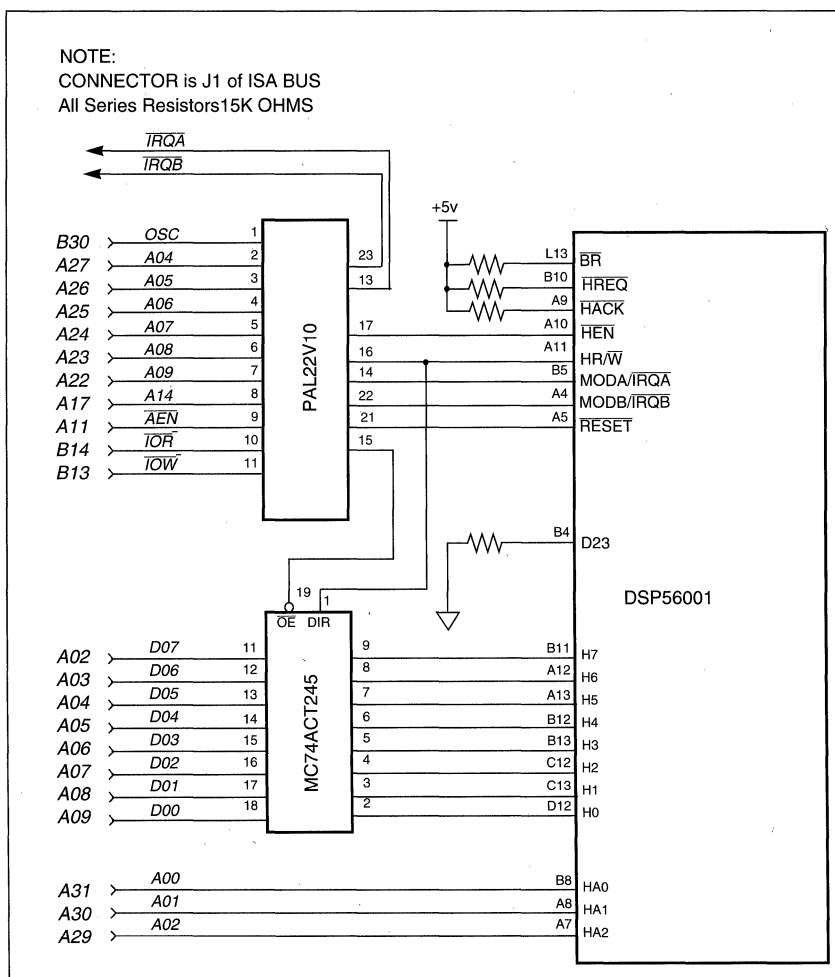


Fig. A.14 DSP56001-to-ISA Bus Interface Schematic illustrates how simple the circuitry is to connect to the ISA bus.

Fig. A.15 PLD Definition for the ISA Bus Interface for the PAL22V10 shown in Figure A.14

```

0001e  module          pcio2
0002e  title            'ISA (IBM-PC) Interface Ver.2
0003e  MOTOROLA INC.   14 February 1991'
0004e
0005e  U01             device 'P22V10';
0006e
0007e  "INPUTS"
0008e  CLK             pin      1;      "ISA-Bus Clock "
0009e  AEN             pin      9;      "Address Enable -- NOT DMA cycle"
0010e  A14,A9,A8       pin      8,7,6;  "ADDRESS Bits14, 09-08"
0011e  A7,A6,A5,A4     pin      5,4,3,2; "ADDRESS Bits 07-04"
0012e  IOR,IOW        pin      10,11;  "I/O Read*,I/O Write*"
0013e  IRQA,IRQB      pin      13,23;
0014e
0015e  "OUTPUTS"
0016e  MODA,MODB       pin      14,22;
0017e  RESET          pin      21;      "Reset* latched"
0018e  Q2,Q1,Q0       pin      20,19,18;
0019e  HEN            pin      17;      "HOST ENABLE*"
0020e  HRw           pin      16;      "HOST R/W*"
0021e  Ben           pin      15;      "Buffer Enable for 74AC245"
0022e
0023e  RESET          ISTYPE      'reg_D,Buffer';
0024e  Q2,Q1,Q0       ISTYPE      'reg_D,Buffer';
0025e  HRw           ISTYPE      'reg_D,Buffer';
0026e
0027e  High,Low       = 1,0;
0028e  H,L,C,K,X     = 1,0,..C...K...X.;
0029e
0030e  StateReset      = [RESET];
0031e  Normal          = [ 1 ];
0032e  SetReset        = [ 0 ];
0033e
0034e  StateDir        = [ HRw ]; "Host Read/WRITE*, buffer direction"
0035e  ReadDir         = [ 1 ];
0036e  WritDir         = [ 0 ];
0037e
0038e  StateNo         = [ Q2..Q0 ];
0039e  Idle            = [ 0,0,0 ];
0040e  S1              = [ 0,0,1 ];
0041e  S2              = [ 0,1,0 ];
0042e  S3              = [ 0,1,1 ];
0043e  S4              = [ 1,0,0 ];
0044e  S5              = [ 1,0,1 ];
0045e  S6              = [ 1,1,0 ];
0046e  S7              = [ 1,1,1 ];
0047e
0048e  Rcyc            = !AEN & A9 & A8 & !A7 & A6 & !A5 & !A4 & !IOR;
0049e  Wcyc            = !AEN & A9 & A8 & !A7 & A6 & !A5 & !A4 & !IOW;
0050e  Addr            = [AEN,A14,A9,A8,A7,A6,A5,A4];
0051e
0052e  state_diagram StateReset
0053e  state Normal:   if (Wcyc & A14) THEN SetReset
0054e                  ELSE      Normal;
0055e  state SetReset: if (Wcyc & !A14) THEN Normal
0056e                  ELSE      SetReset;
0057e
0058e  state_diagram StateDir
0059e  state ReadDir:  if (!IOW) THEN WritDir
0060e                  ELSE      ReadDir;
0061e  state WritDir:  if (!IOR) THEN ReadDir
0062e                  ELSE      WritDir;
0063e
0055e  state SetReset: if (Wcyc & !A14) THEN Normal
0056e                  ELSE      SetReset;
0057e
0058e  state_diagram StateDir
0059e  state ReadDir:  if (!IOW) THEN WritDir

```

```

0060e      ELSE      ReadDir;
0061e      state WritDir: if (I/O) THEN ReadDir
0062e      ELSE      WritDir;
0063e
0064e state_diagram StateNo
0065e   State Idle: HEN = 1; Ben = 1;
0066e       if (!Rcyc & !Wcyc) THEN Idle "stay put if not for me..."
0067e       ELSE S1;
0068e State S1: HEN = 1; Ben = 1; goto S2;"allow time to select 74AC245 direction"
0069e   State S2: HEN = 1; Ben = 0; goto S3; "Now, enable the 74AC245 output"
0070e   State S3: HEN = 0; Ben = 0; "assert HEN*"
0071e       if (Rcyc # Wcyc) THEN S3 "...and loop until the end of the I/O cycle"
0072e       ELSE S4;
0073e   State S4: HEN = 1; Ben = 0; goto Idle; "deassert HEN*, and quit"
0074e   State S5: goto Idle; "these are dummies, just in case.."
0075e   State S6: goto Idle;
0076e   State S7: goto Idle;
0077e equations
0078e   [Q2,Q1,Q0].ck = CLK;
0079e   RESET.ck = CLK;
0080e   HRw.ck = CLK;
0081e
0082e   MODA = !(RESET.Q & !IRQA);
0083e   MODB = RESET.Q & IRQB;
0084e Test_vectors
0085e   ([CLK,Addr,IOW,IOR,IRQA,IRQB] -> [HRw,RESET,MODA,MODB])
0086e   [ C.^h34,0,1,0,1] -> [0,1,0,1]; "write to port, sets write dir."
0087e   [ C.^h74,0,1,0,1] -> [0,0,1,0]; "write to reset address, asserts reset"
0088e   [ C.^h34,1,0,0,1] -> [1,0,1,0]; "read from normal address, reset "
0089e   [ C.^h34,0,1,0,1] -> [0,1,0,1]; "write to normal address, deasserts reset"
0090e   [ C.^h34,1,1,0,1] -> [0,1,0,1];
0091e   [ C.^h34,1,1,1,1] -> [0,1,1,1];
0092e Test_vectors
0093e   ([CLK,Addr,IOW,IOR] -> [HEN,Ben,HRw])
0094e   [C.^h24,0,1] -> [1,1,0]; "this is NOT for me...., wrong addr"
0095e   [C.^h34,1,1] -> [1,1,0]; "cycle starts, addresses valid..."
0096e   [C.^h34,1,1] -> [1,1,0];
0097e   [C.^h34,1,0] -> [1,1,1]; "read cycle identified by IOR* "
0098e   [C.^h34,1,0] -> [1,0,1];
0099e   [C.^h34,1,0] -> [0,0,1];
0100e   [C.^h34,1,0] -> [0,0,1];
0101e   [C.^h34,1,0] -> [0,0,1];
0102e   [C.^h34,1,1] -> [1,0,1];
0103e   [C.^h34,1,1] -> [1,1,1];
0104e   [C.^h34,0,1] -> [1,1,0];
0105e
0106e END pcio2

```

A.5.2 Detailed Circuit Description

The ISA bus delineates two types of bus accesses — memory and I/O. The distinction is made by the use of separate read and write strobes for each type of access. The interface in this example is mapped into the ISA Bus processor's I/O space in the address range \$340-\$34F.

In order to provide a facility for bootstrap initiation, the RESET pin of the DSP56001 is driven by a latch which is mapped into the host I/O space. Writes to any I/O address in the range \$348-\$34F will assert RESET to the DSP56001. A write to any address within the range \$340-\$347 will deassert the RESET latch.

The 8 registers which comprise the DSP56001 Host Interface are mapped into the ISA bus I/O space between address \$340-\$347 (inclusive). Communications with the DSP56001, including program bootstrapping, are accomplished via I/O reads and I/O writes to the appropriate register. Please refer to the **DSP56001 User's**

Manual, especially chapter 10, for a detailed description of the Host Interface and its usage.

The bootstrap mode on the DSP56001 is selected via the processor's MODA and MODB inputs. The PLD provides the proper logic levels on these lines during reset. After reset, the MODA and MODB inputs reflect the state of the interrupt request inputs $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$, thus permitting the normal use of the external interrupt structure of the DSP56001 without forcing constraints on the behavior of the interrupt lines during reset.

During a transfer cycle to/from the Host Interface Registers, the PLD functions as a simple state machine which sequences the control signals for the bus transceiver and the data strobe. Because the ISA bus I/O cycles have relatively long periods, slower PLDs often prove adequate. When attaching this interface to 33MHz machines, a 15ns PLD is recommended.

A.5.3 Timing

Figure A.16 depicts the timing relationships present during ISA Bus I/O Read cycles and I/O Write cycles. The duty cycle of the processor clock has a 2:1 ratio of low period to high period and a frequency of one-third that of the master oscillator. During an I/O cycle either $\overline{\text{IOW}}$ or $\overline{\text{IOR}}$ will be asserted.

The most critical aspect of this interface is the relationship between Host Enable ($\overline{\text{HEN}}$) and the other interface signals. The Host address lines, HA0-2 and Host Read/Write (HR/ $\overline{\text{W}}$) must remain stable during the period in which $\overline{\text{HEN}}$ is asserted. The propagation delays associated with the MC74ACT245 transceiver have been considered and the complications typical of asynchronous interfaces have been avoided by running the PLD clock from the system oscillator which operates at 3x the processor clock. During successive cycles of the oscillator, the transceiver's direction is established, the transceiver is enabled, and $\overline{\text{HEN}}$ is strobed. The ISA bus indicates the completion of the data transfer by releasing $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$ (as appropriate to the direction of transfer) and the PLD deasserts $\overline{\text{HEN}}$ on the following oscillator cycle. The MC74ACT245 is disabled on the device enable.

Please refer to the *DSP56001 User's Manual*, Motorola Document DSP56001/D, for a complete description of the operation of the DSP56001.

A.6 COMMUNICATE WITH THE DSP56000 HOST INTERFACE USING C LANGUAGE

"The C language source code and the source for the PLD used in the hardware interface are both available on Motorola's Dr. BuB BBS."

A.6.1 Introduction

Interfacing a DSP56000/1/2 target system to an ISA-bus is only partially complete when the hardware is in place. Download software is the other element required

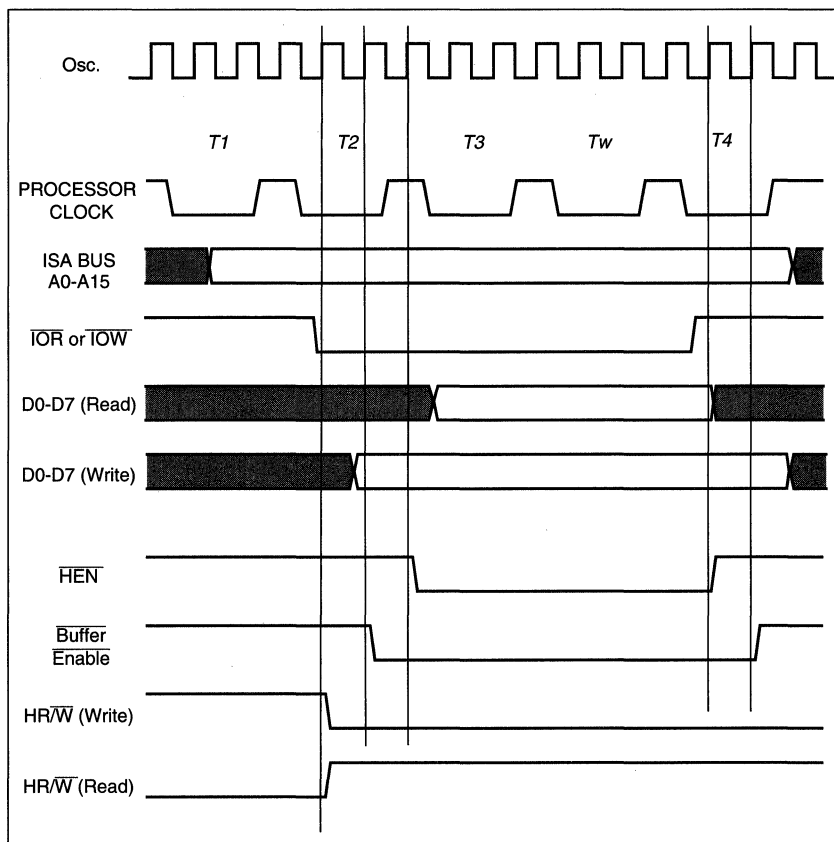


Fig. A.16 DSP56001-to-ISA Bus Interface Timing shows the timing for both a read and a write operation.

before debugging begins. The specific target hardware determines the types of tasks relegated to the download software. We assume that the user has a target system similar to the DSP-to-ISA interface described in Figure A.14.

A.6.2 Example Program

The following example uses the DSP56002. The download task can be subdivided into four steps:

- reset the DSP
- verify that the DSP is present (at the expected location)
- transfer code into the DSP's internal P:RAM terminate the boot
- execute the loaded application

Assume that the DSP56002's target host interface (HI) registers have been mapped by the interface hardware into the ISA-bus I/O address space from 0x340 through 0x347. Refer to the **DSP56002 User's Manual, SECTION 5** for a description of the HI register set visible to the host processor. Additionally, the target hardware has a latch attached to the DSP56002 RESET* which can be set (RESET* asserted) by ISA-bus writes to 0x350 (the data is ignored) and cleared by ISA-bus reads from the same address.

The C language program which appears in Figure A.17 performs all of the download tasks listed above. After some initialization of the screen, the code jumps to the download routine "RESET56." This routine resets the DSP and waits for a short delay to assure that the device receives a reset pulse of adequate duration. Following this, the reset is released and another delay is invoked simply to provide time for the target system to recognize the release of reset and to start executing the routine in its internal bootstrap ROM. This routine will:

- ☞ sample the mode selection lines (MODA/ $\overline{\text{IRQA}}$, MODB/ $\overline{\text{IRQB}}$, MODC/ $\overline{\text{NMI}}$) to determine the type of boot desired (from a target processor via the HI in this case)
- ☞ branch to the ROM code which will initialize the DSP56002 as required (for booting from the HI)
- ☞ start receiving code from the selected bootstrap peripheral (the HI)
- ☞ jump to the start of the newly downloaded code upon completion of the boot.

After the reset sequence, the download routine which is running on the ISA-bus host will check for the presence of a functional DSP56002 host interface by reading four of the HI registers and by checking for presence of the default values. This is a good check of the DSP56002 reset sequence (if it does not reset properly and detect the desired boot mode, the HI will not be enabled) and of the interface hardware's ability to read the HI. If the proper default values are not sensed, the program exits and returns to the command line prompt.

If a DSP56002 is found at the expected address, the ISA download program proceeds to load the initial target code into the DSP's internal P:RAM at addresses P:\$0000-01FF. Recall that this process occurs while the DSP56002 on-chip PLL is set to multiply the external oscillator frequency by one, so it can be a (relatively) slow process if the DSP is being run from a slow external clock. To save time when loading DSP routines which do not require the entire on-chip P:RAM space, the boot can be terminated early by setting HOST FLAG bit 0 (as is shown in the listing in Figure A.17). For brevity, the actual code to be downloaded is present in the example as a statically declared buffer. The user may prefer to write a function to place .LOD or .CLD formatted disk data into a buffer which is passed to the download function.

This example should serve as a beginning for a host download capability through the DSP56000/1/2 Host Interface. The C language source code and the source for the PLD used in the hardware interface are both available on Motorola's Dr. BuB BBS.

Fig. A.17 Example Program of DSP56000 Host Interface Using C Language

```

/* hostio.c - host I/F test                               */
/* compiled with Tubro-C version 2.01                       */
/*      20 May 1992                                         */

#include <stdio.h>
#include <dos.h>
#include <process.h>

#define RSTADDR      0x0350 /* ISA-bus address of RESET latch */
#define BASE         0x0340 /* ISA-bus address of Host Interface*/
#define ICR          BASE
#define CVR          BASE+1
#define ISR          BASE+2
#define IVR          BASE+3
#define RXH          BASE+5
#define RXM          BASE+6
#define RXL          BASE+7
#define TXH          BASE+5
#define TXM          BASE+6
#define TXL          BASE+7

#define DELAY        10000
#define BOOTSIZE1    7
#define BOOTSIZE2    9

int reset56 (int, unsigned char *);

/*****
/* simple code to send an incrementing pattern of bytes to host */
*****/
/*          org      p:$0                                         */
/*          begin                                         */

unsigned char BOOT1[] =
{
    0x08,0xF4,0xA0, /* movep#$0001,x:$FFE0 */
    0x00,0x00,0x01, /* */
    0x08,0xC8,0x2B, /* movep  A0,x:$FEEB */
    0x00,0x00,0x08, /* inc    A */
    0x0A,0xA9,0x81, /* jclr   #1,x:$FEE9,* */
    0x00,0x00,0x04, /* */
    0x0C,0x00,0x00}; /* jmp    <begin */

void main()
{
    unsigned char i,j;
    int k;

    system("cls");

/*****
/* boot the 56002 with the patterns-to-host routine */
*****/
    printf("\n BOOTING 56002-Incrementing Patterns to Host");
    if (reset56(BOOTSIZE1,BOOT1) == -1)
        exit(-1);
    printf("\n TESTING DATA READ (56002-to-HOST)\n");
    k = 0xFFFF;
    outportb (ICR,0); /* assure proper mode, DMA off, etc. */
    while ( (inportb(ISR) & 0x01) == 0); /* wait for RXDF == 1 */
    j = (inportb(RXL) & 0xFF); /* get first pattern */
    do {
        while ( (inportb(ISR) & 0x01) == 0); /* wait for RXDF == 1 */
        i = (inportb(RXL) & 0xFF); /* get next pattern */
        if (i != ((j+1) & 0xFF)) /* check and advise */
            printf("\n ERROR! Rcvd: %2X Expected: %2X",i,j);
        j = i; /* re-sync pattern */
    } while (k--);
    printf("\n READ TEST COMPLETE\n\n");
}

```

```

/*****
/* routine to reset and boot the DSP56001/2
*****/
int reset56( int codesize, unsigned char *codeptr)
{
    unsigned char    icr_rd, cvr_rd, isr_rd, ivr_rd;
    int              i, j, k;

/* first, assert reset. Leave reset asserted for a while before */
/* releasing it...give the DSP time to exit reset, sample the */
/* MODA, MODB, MODC pins and start the bootstrap code...      */

    outp ((int)RSTADDR, 0);          /* reset the DSP */
    for (k=0; k<DELAY; k++);        /* wait */
    inportb (RSTADDR);              /* clear reset */
    for (k=0; k<DELAY; k++);        /* wait again */
    printf("\nRESET CYCLED\n");      /* eye candy */

/* then, verify that DSP56002 is present: read the icr, cvr, isr, ivr */
/* ...and look for default values in these registers. Refer to the */
/* DSP56000/1/2 Family User's Manual for a complete description of */
/* Host Interface (Port-B) and its register set.                  */

    icr_rd = inportb(ICR);          /* Interrupt Control Register s/b 0x00 */
    cvr_rd = inportb(CVR);          /* Command Vector Register s/b 0x12 */
    isr_rd = inportb(ISR);          /* Interrupt Status Register s/b 0x06 */
    ivr_rd = inportb(IVR);          /* Interrupt Vector Register s/b 0x0F */

    printf
    (" \n HOST I/F RETURNED:  ICR: %2X  IVR: %2X  ISR: %2X  IVR: %2X \a",
     icr_rd, cvr_rd, isr_rd, ivr_rd);

    if ((icr_rd != 0x00) || (cvr_rd != 0x12)
        || (isr_rd != 0x06) || (ivr_rd != 0x0F))
    {
        /* if default values */
        printf("\n \a RESET FAILED! "); /* NOT found, advise */
        return(-1);                    /* and return error */
    }
    else
    {
        /* BOOT THE DSP */
        for (i=0, j=0; i<codesize; i++) /* NOTE: always send */
        {                               /* the lsbyte last */
            while ( (inportb(ISR) & 0x02) != 2); /* wait for TXDE == 1 */
            outportb(TXH, codeptr[j++]); /* send upper byte */
            outportb(TXM, codeptr[j++]); /* send middle byte */
            outportb(TXL, codeptr[j++]); /* send low byte */
        }
        outportb(ICR, 0x08);          /* terminate Host Boot */
        return(0);
    }
}

```

A.7 REFERENCES

1. Eggebrecht, Lewis, *Interfacing to the IBM Personal Computer*, Howard W. Sams & Company, 1988.
2. *DRAM Refresh Modes*, Motorola Application Note AN987.
3. Motorola FACT Data, DL138, Rev.1.
4. Motorola's DSP56000/DSP56001 Digital Signal Processor User's Manual, DSP56000UM/AD, Rev.2.
5. Motorola's DSP56001 56-Bit General Purpose Digital Signal Processor, Advance Information DSP56001/D, Rev.1.

6. Motorola's MCM514256A Data sheet, Motorola Memory Data, DL113, Rev.5, pp.84-98.
7. *Page, Nibble, and Static Column Modes...*, Motorola Application Note AN986.
8. PAL[®] Device Handbook, Advanced Micro Devices/Monolithic Memories Inc., 1988.
9. PAL[®] Devices Databook, Advanced Micro Devices, 1990.

Implementing IIR/FIR Filters with Motorola's DSP56000/ DSP6001

Please note that the DSP56001 is obsolete. It has been replaced by the DSP56001A/56002, depending on degree of compatibility required.

B.1 INTRODUCTION

"Two classes of frequency-selective digital filters are considered: infinite impulse response (IIR) and finite impulse response (FIR) filters."

This application note considers the design of frequency-selective filters, which modify the frequency content and phase of input signals according to some specification. Two classes of frequency-selective digital filters are considered: infinite impulse response (IIR) and finite impulse response (FIR) filters. The design process consists of determining the coefficients of the IIR or FIR filters, which results in the desired magnitude and phase response being closely approximated.

Therefore, this application note has a two-fold purpose:

1. to provide some intuitive insight into digital filters, particularly how the coefficients are calculated in the digital domain so that a desired frequency response is obtained, and
2. to show how to implement both classes of digital filters (IIR and FIR) on the DSP56001.

It is assumed that most readers are analog designers learning digital signal processing (DSP). The approach used reflects this assumption in that digital filters are initially presented from an analog point of view. Hopefully, this approach will simplify

the transition from the analog s-domain transfer functions to the equivalent functions in the digital z-domain. In keeping with this analog perspective, IIR filters will be discussed first since the equivalent of FIR filters are infrequently encountered in the analog world.

SECTION B.1 is a brief review of lowpass, highpass, bandpass, and bandstop analog filters. The s-domain formulas governing the key characteristics, magnitude-frequency response, $G(\Omega)$, and phase-frequency response, $\phi(\Omega)$, are derived from first principles. Damping factor, d , cutoff frequency, Ω_c , for lowpass and highpass filters, center frequency, Ω_0 , for bandpass and bandstop filters, and quality factor, Q , are defined for the various filter types.

SECTION B.2 introduces the bilinear transformation so that analog s-domain designs can be transformed into the digital z-domain and the correct coefficients thereby determined. The form of the formulas for the z-domain filter coefficients thus determined are generalized in terms of the key filter characteristics in the z-domain so that the engineer can design digital filters directly without the necessity of designing the analog equivalent and transforming the design back into the digital domain.

In the analog domain, the performance of the filter depends on the tolerance of the components. Similarly, in the digital domain, the filter performance is limited by the precision of the arithmetic used to implement the filters. In particular, the performance of digital filters is extremely sensitive to overflow, which occurs when the accumulator width is insufficient to represent all the bits resulting from many consecutive additions. This condition is similar to the condition in the analog world in which the signal output is larger than the amplifier power supply so that saturation occurs. A short analysis of the gain at critical nodes in the filters is given in **SECTION B.3** and **SECTION B.4** to provide some insight into the scaling requirements for different forms of IIR filters. For this reason, the signal flow graphs developed are centralized about the accumulator nodes.

The analysis of IIR filters focuses on second-order sections. Clearly, higher order filters are often required. Therefore, a brief discussion of how second order sections can be cascaded to yield higher order filters is given in **SECTION B.5**. Because the analysis becomes complex quickly, the discussion naturally leads to using commercially available filter design software such as Filter Design and Analysis System (FDAS) from Momentum Data Systems, Inc. **SECTION B.6** concludes by showing how the filter coefficients just discussed can be used in DSP56000 code to implement practical digital filters. Examples of complete filter designs are given, including the code, coefficients, frequency response, and maximum sample frequency.

FIR filters are discussed in **SECTION B.7**. Initially, FIR filters are contrasted with IIR filters to show that in many ways they are complementary, each satisfying weaknesses of the other. An intuitive approach is taken to calculating the filter coefficients by starting from a desired arbitrary frequency response. The importance of and constraint imposed by linear phase is emphasized. Having developed an intuitive appreciation of what FIR filter coefficients are, the use of FDAS to accelerate the design process is described. **SECTION B.7** concludes by showing how the filter coefficients just determined can be used in DSP56000 code to implement practical digital filters. An example of a passband digital filter using a Kaiser-window design approach is presented.

B.1.1 Analog RCL Filter Types

In the following paragraphs, the analog RCL filter network will be analyzed for the four basic filter types: lowpass, highpass, bandpass, and bandstop. Analyzing analog filter types shows that designing digital IIR filters is, in many cases, much simpler than designing analog filters.

In this analysis, as in all of the following cases, the input is assumed to be a steady-state signal containing a linear combination of sinusoidal components whose rms (or peak) amplitudes are constant in time. This assumption allows simple analytic techniques to be used in determining the network response. Even though these results will then be applied to real-world signals that may not satisfy the original steady-state assumption, the deviation of the actual response from the predicted response is small enough to neglect in most cases. General analysis techniques consist of a linear combination of steady-state and transient response solutions to the differential equations describing the network.

B.1.2 Analog Lowpass Filter

The passive RCL circuit forming a lowpass filter network is shown in Figure B.1 where the transfer function, $H(s)$, is derived from a voltage divider analysis of the RCL network. This approach is valid since the effect of C and L can be described as a complex impedance (or reactance, X_C and X_L) under steady state conditions; s is a complex variable of the complex transfer function, $H(s)$. The filter frequency response is found by evaluating $H(s)$ with $s = j\Omega$, where $\Omega = 2\pi f$ and f is the frequency of a sinusoidal component of the input signal. The output signal is calculated from the product of the input signal and $H(j\Omega)$. To facilitate analysis, the input and output signal components are described by the complex value, $e^{j\Omega t} = \cos \Omega t + j \sin \Omega t$. The actual physical input and output signal components are found by taking the real part of this value. The input is $R\{e^{j\Omega t}\} = \cos \Omega t$; the output is $R\{H(j\Omega)e^{j\Omega t}\} = G(\Omega) \cos [\Omega t + \phi(\Omega)]$. The previous technique is based upon the solution of the differential equations describing the network when the input is steady state. Describing the circuit response by $H(s)$ instead of solving the differential equation is a common simplification used in this type of analysis.

The magnitude of $H(s)$ is defined as the gain, $G(\Omega)$, of the system; whereas, the ratio of the imaginary part to real part of $H(s)$, $I\{H(j\Omega)\}/R\{H(j\Omega)\}$, is the tangent of the phase, $\phi(\Omega)$, introduced by the filter. If the input signal is $A_k \sin (\Omega_k t + \phi_k)$, then the output signal is $A_k G(\Omega_k) \sin [\Omega_k t + \phi_k + \phi(\Omega_k)]$. Figure B.2 shows the gain, $G(\Omega)$, and phase, $\phi(\Omega)$, plots for the second-order lowpass network of Figure B.1 for various values of damping factor, d ; d also controls the amplitude and position of the peak of the normalized response curve.

The frequency corresponding to the peak amplitude can be easily found by taking the derivative of $G(\Omega)$ (from the equation for $G(\Omega)$ in Figure B.1) with respect to Ω and setting it equal to zero. Solving the resultant equation for Ω then defines Ω_M as the frequency where the peak amplitude occurs. The peak amplitude is then $G_M = G(\Omega_M)$:

$$\Omega_M = \Omega_c \sqrt{(1 - d^2/2)} \quad (\text{B.1})$$

$$G_M = \frac{1}{d \sqrt{(1 - d^2/4)}} \quad (\text{B.2})$$

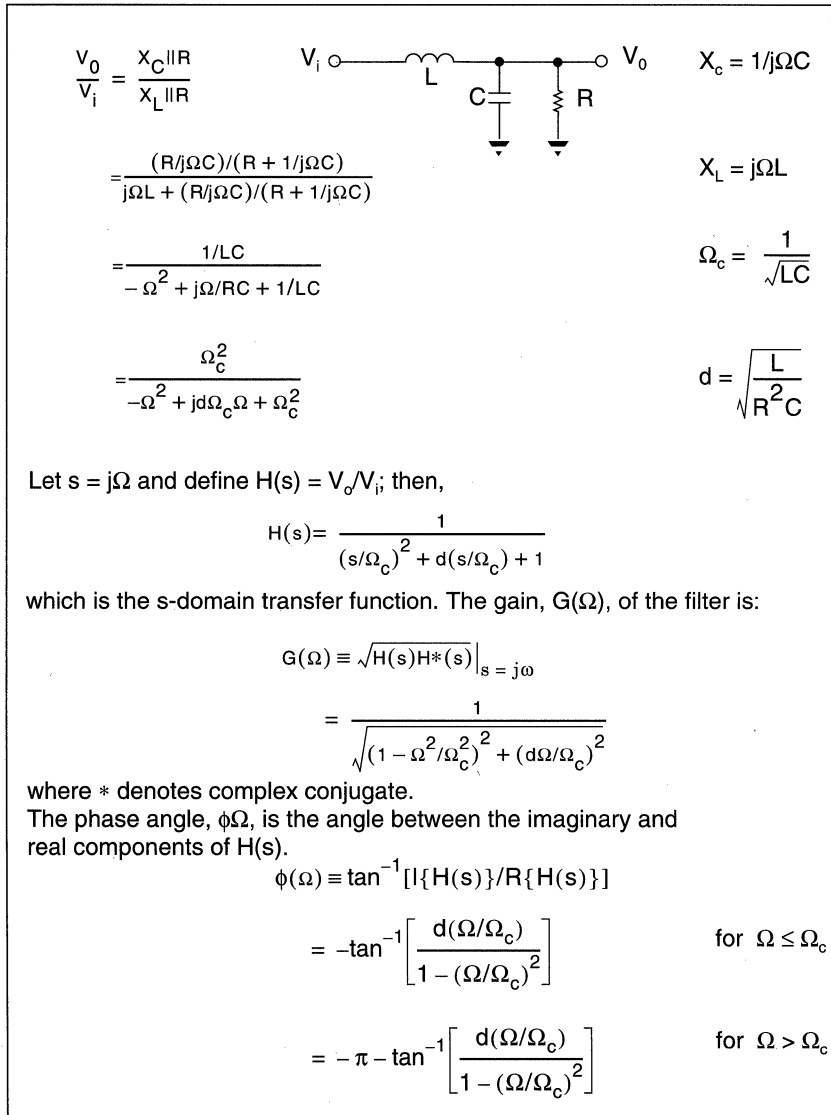


Fig. B.1 s-Domain Analysis of Second-Order Lowpass Analog Filter

for $d < \sqrt{2}$. For $d > \sqrt{2}$, $\Omega_M = 0$ is the position of the peak amplitude where $G_M = 1$. When $d = \sqrt{2}$, $G_M = 1$, which gives the maximally flat response curve used in the Butterworth filter design (usually applies only to a set of cascaded sections). Note that Ω_C for a lowpass filter is that frequency where the gain is $G(\Omega_C) = 1/d$ and the phase is $\phi(\Omega_C) = -\pi/2$.

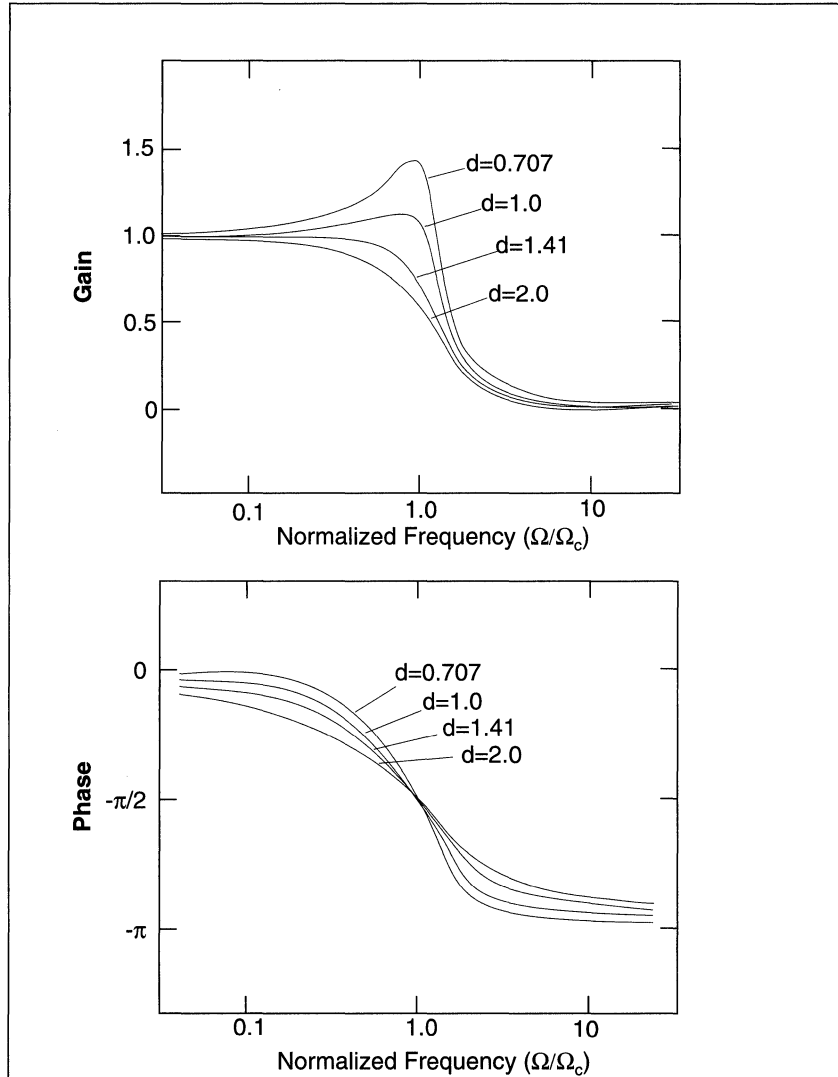


Fig. B.2 Gain and Phase Response of Second-Order Lowpass Analog Filter at Various Values of Damping Factor, d

B.1.3 Analog Highpass Filter

The passive RCL circuit forming a highpass filter network is shown in Figure B.3 where the transfer function, $H(s)$, is again derived from a voltage divider analysis of the RCL network. The gain and phase response are plotted in Figure B.4 for different values of damping coefficient. As evidenced, the highpass filter response is the mirror image of the lowpass filter response.

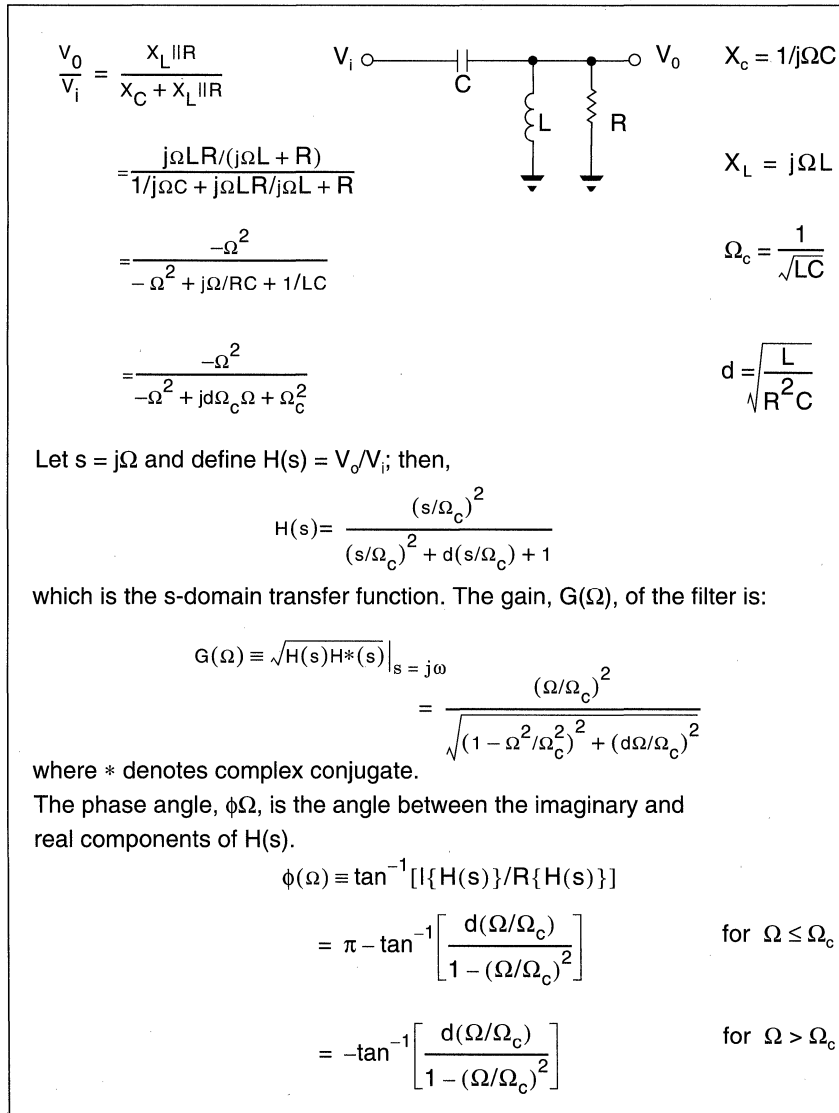


Fig. B.3 s-Domain Analysis of Second-Order Highpass Analog Filter

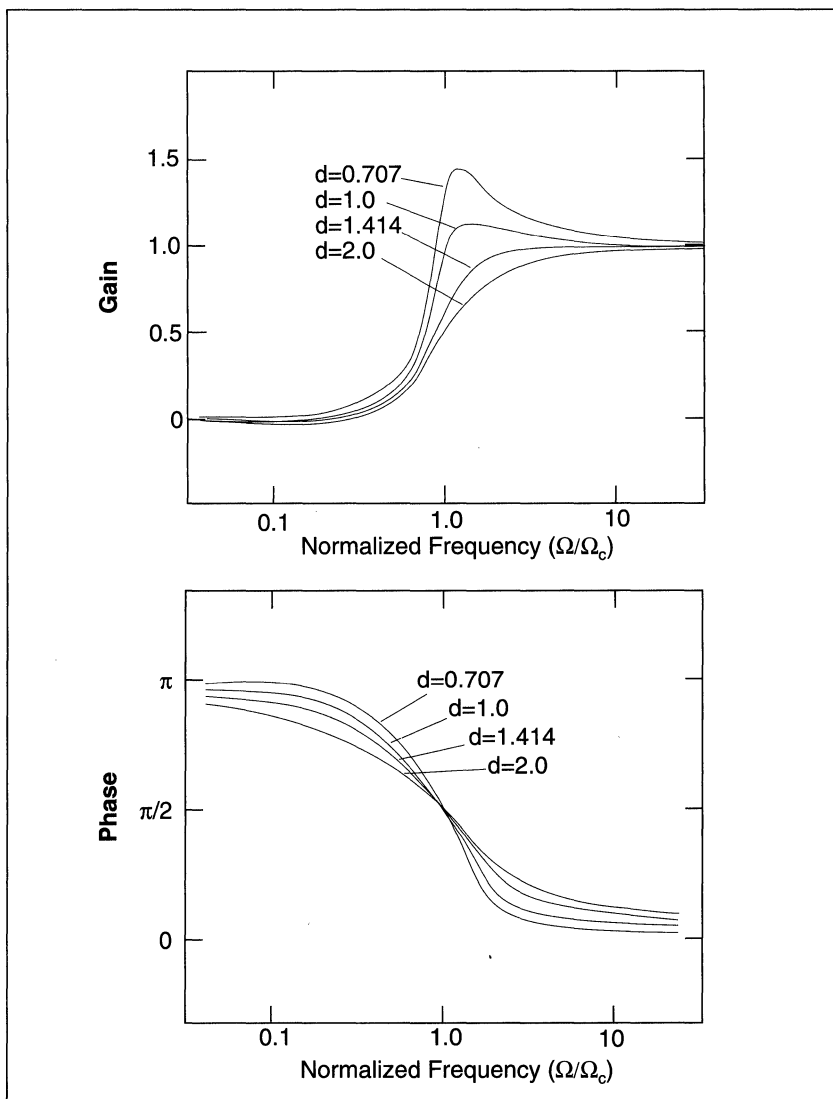


Fig. B.4 Gain and Phase Response of Second-Order Highpass Analog Filter at Various Values of Damping Factor, d

B.1.4 Analog Bandstop Filter

The analog RCL network for a bandstop filter network is simply the sum of the lowpass and highpass transfer functions shown in Figure B.5 where the transfer function, $H(s)$, is again derived from a voltage divider analysis of the RCL network. The gain and phase response are plotted in Figure B.6 for different values of quality factor,

Q , (where $Q = 1/d$). Neglecting the departure of real RCL components' values from the ideal case, the attenuation at the center frequency, f_0 , is infinite. Also, note that the phase undergoes a 180-degree shift when passing through the center frequency (zero in the s -plane).

Q for bandpass and bandstop filters is a measure of the width, $\Delta\Omega$, of the stop-band with respect to the center frequency, Ω_0 , i.e., $\Delta\Omega = Q^{-1}\Omega_0$. $\Delta\Omega$ is measured at the points where $G(\Omega) = 1/\sqrt{2}$.

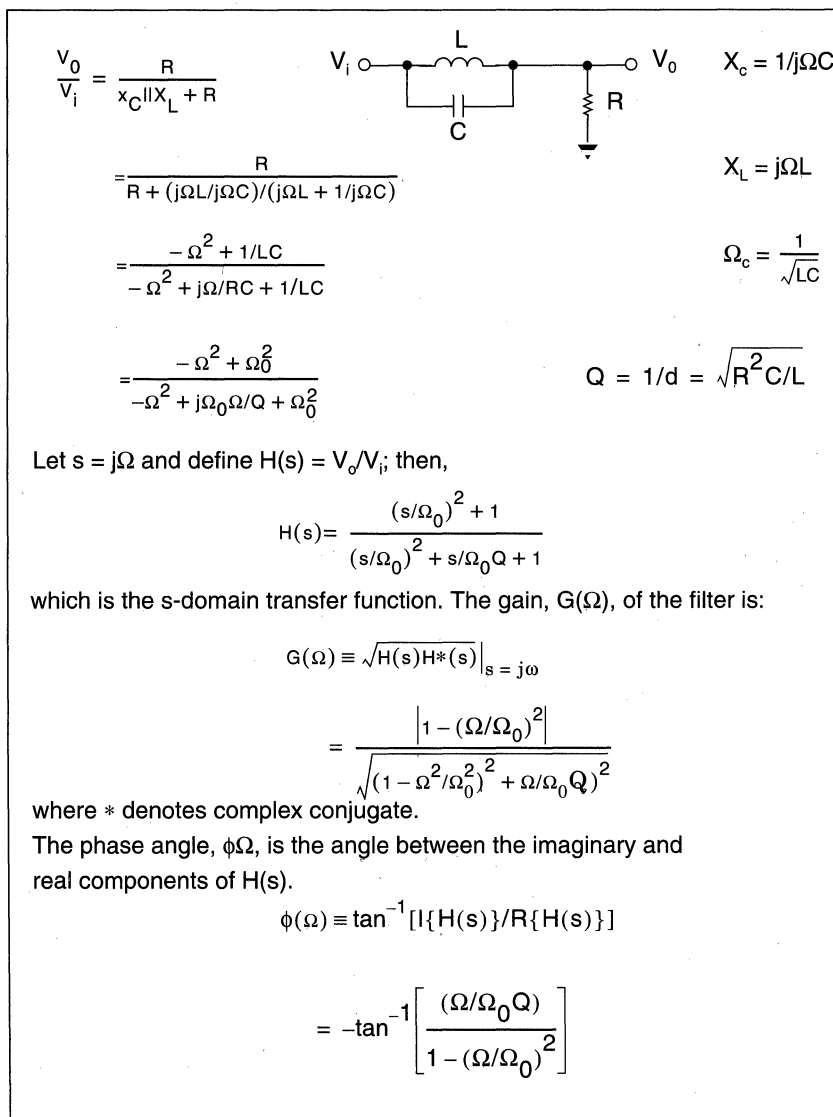


Fig. B.5 s-Domain Analysis of Second-Order Bandstop Analog Filter

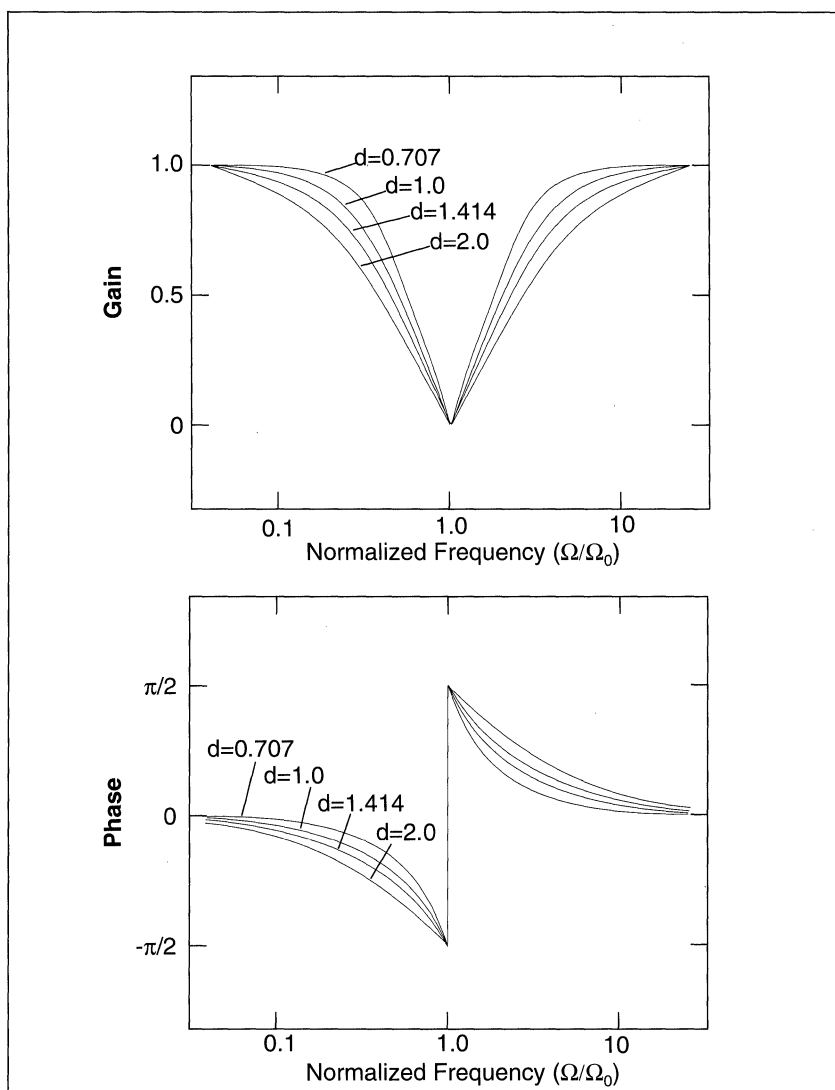


Fig. B.6 Gain and Phase Response of Second-Order Bandstop Analog Filter at Various Values of Damping Factor, d

B.1.5 Analog Bandpass Filter

The passive RCL circuit forming a bandpass filter network is shown in Figure B.7 where the transfer function, $H(s)$, is again derived from a voltage divider analysis of the RCL network. The gain and phase response are plotted in Figure B.8 for different values of Q . The lowpass gain approaches an asymptotic function of $G = (f_c/f)^2$ for $f/$

$f_0 \gg 1$. The highpass asymptotic gain is $G = (f_c/f)^2$ for $f/f_c \ll 1$; whereas, the bandstop case approaches unity at zero and infinity with a true zero at the center frequency. The bandpass gain, on the other hand, approaches $G = Q^{-1}f_c$ for $f/f_c \ll 1$ and $G = Q^{-1}f_c/f$ for $f/f_c \gg 1$. The primary differences to note in the bandpass response are:

1. the stopband attenuation is 6 dB/octave or 20 dB/decade (since it goes as $1/f$); whereas, the lowpass and highpass go as $1/f^2$ (12 dB/octave, 40 dB/decade)
2. the stopband attenuation asymptote is dependent on the quality factor; whereas,

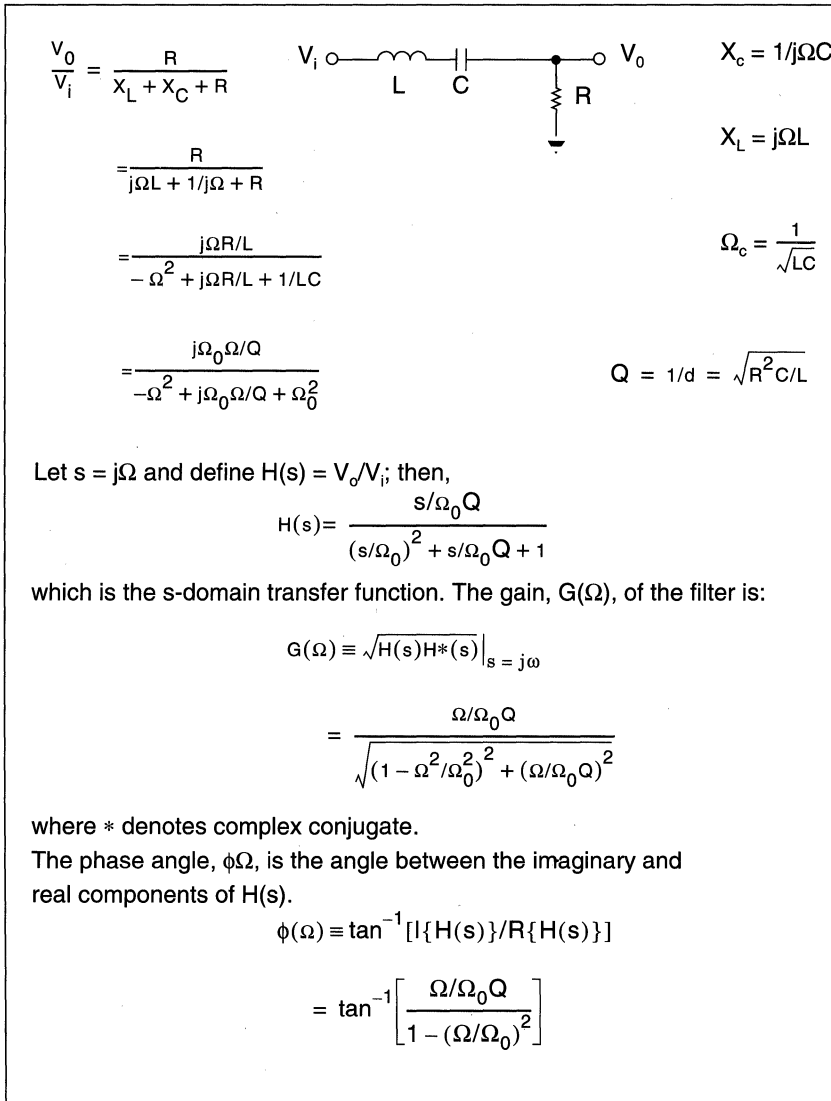


Fig. B.7 s-Domain Analysis of Second-Order Bandpass Analog Filter

for the lowpass and highpass cases, the stopband attenuation asymptote is independent of damping factor, d

3. the maximum value of gain is unity regardless of the filter Q

The specific features characterizing the bandpass, lowpass, highpass, and band-stop analog networks are found to be nearly identical in the digital IIR filter equivalents when the sampling frequency is very high as compared to the frequencies of

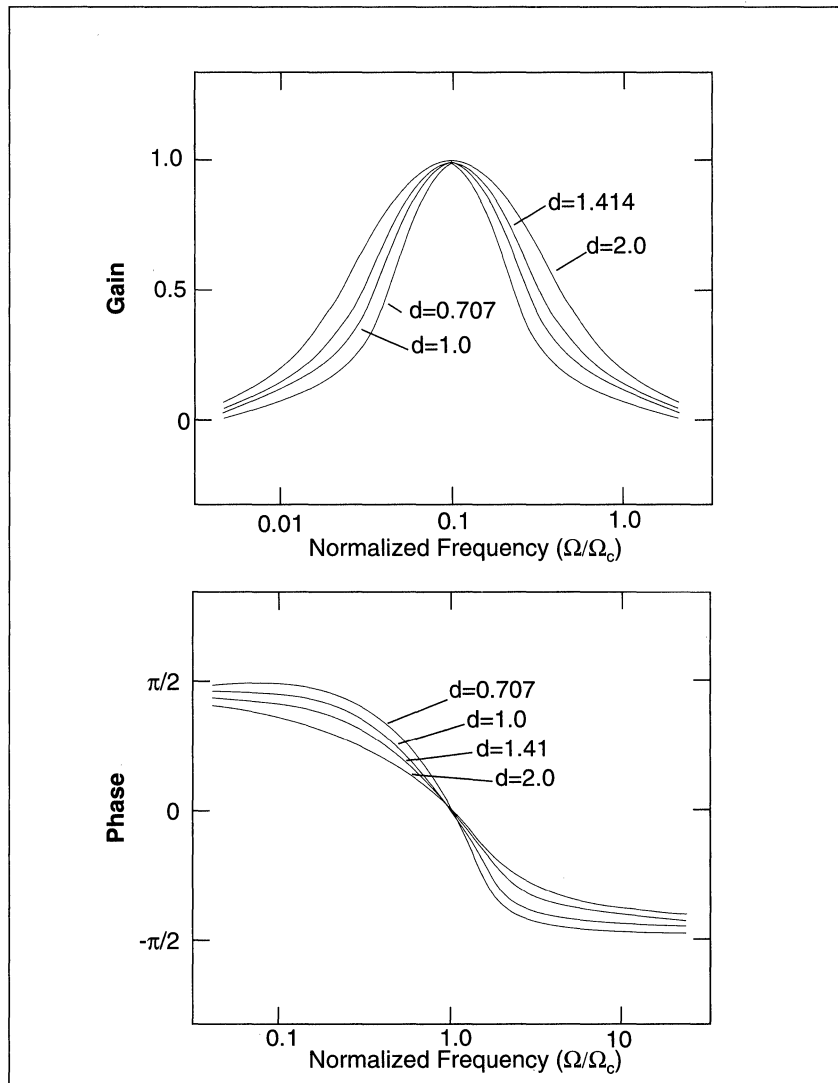


Fig. B.8 Gain and Phase Response of Second-Order Bandpass Analog Filter at Various Values of Damping Factor, d

interest. For this reason, it is important to understand the basic properties of the four filter types before proceeding to the digital domain.

B.2 SECOND-ORDER DIRECT-FORM IIR DIGITAL FILTER SECTIONS

"In the direct-form implementation, the a_i and b_i are used directly in the difference equation, which can be easily programmed on a high-speed DSP such as the DSP56001."

The traditional approach to deriving the digital filter coefficients has been to start with the digital z-domain description, transform to the analog s-domain to understand how to design filters, then transform back to the digital domain to implement the filter. This approach is not used in this report. Instead, formulas are developed relating the s-domain filter to the z-domain filter so transformations to and from one domain to the other are no longer necessary.

The Laplace or s-transform in the analog domain was developed to facilitate the analysis of continuous time signals and systems. For example, using Laplace transforms the concepts of poles and zeros, making system analysis much faster and more systematic. The Laplace transform of a continuous time signal is:

$$X(s) = L\{x(t)\} = \int_0^{\infty} x(t)e^{-st} dt \quad (B.3)$$

where: L = the Laplace transform operator and implies the operation described in Eqn. (B.3)

In the digital domain, the continuous signal, $x(t)$, is first sampled, then quantized by an analog-to-digital (A/D) converter before being processed. That is, the signal is only known at discrete points in time, which are multiples of the sampling interval, $T = 1/f_s$, where f_s is the sampling frequency. Because of the sampled characteristic of a digital signal, its z-transform is given by a summation (as opposed to an integral):

$$X(z) = Z\{x(n)\} = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \quad (B.4)$$

where: Z is the z-transform operator as described by the operation of Eqn. (B.4)

$x(n)$ is the quantized values from the A/D converter of the continuous time signal, $x(t)$, at discrete times, $t = nT$

One property of the z-transform which will be used later in this report is the time shifting property. The time shifting property states:

$$x(n-k) = Z^{-1}\left\{z^{-k}X(z)\right\} \quad (B.5)$$

The proof of this property follows directly from the definition of the z-transform. An obvious question arises: "If the s-domain of a signal, $x(t)$, is known and that same signal is digitized, what is the relationship between the s-domain transform and the z-transform?" The relationship or mapping is not unique and depends on the viewpoint used. It is obvious that the trivial mapping, $s = z$, is inappropriate; the signal has been sampled. When a bandlimited signal is sampled (i.e., multiplied by a periodic impulse function), the spectrum of the resulting signal is repetitive as shown in Figure B.9 (see Reference 10).

Clearly, the spectrum consists of the spectrum of the bandlimited signal repeated at multiples of the sampling frequency, $\omega_s = 2\pi f_s$. That is, the resulting spectrum is unique only between 0 and $\omega_s/2$ or multiples thereof; whereas, before the signal was sampled, the energy at frequencies greater than those in the signal was independent of the signal. Therefore, acceptable mappings would either reflect the cyclic nature of the spectrum of the sampled test or at least be linear over the frequencies of interest.

One mapping or transformation from the s-domain to the z-domain discussed later in this report is the bilinear transformation. To understand the origin of this transformation, consider the simple first-order linear analog filter with the system function:

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b}{s + a} \quad (\text{B.6})$$

Recall the differentiation property of the s-transform when $x(t) = L^{-1}\{X(s)\}$ (where L^{-1} is the inverse Laplace transform operation); then the time derivative of $x(t)$ is:

$$\frac{d}{dt}x(t) = L^{-1}\{sX(s)\} \quad (\text{B.7})$$

where: $\frac{d}{dt}x(t)$ is the time derivative of $x(t)$

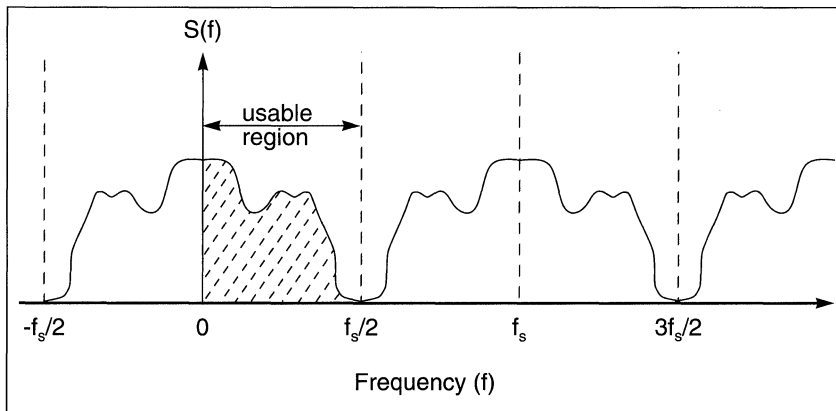


Fig. B.9 Spectrum of Bandlimited Signal Repeated at Multiples of the Sampling Frequency, f_s

Using the differentiation property of Eqn. (B.7), the linear system described by Eqn. (B.6) can be expressed as follows:

$$\frac{d}{dt}y(t) + ay(t) = bx(t) \quad (\text{B.8})$$

If this differential equation is solved by expressing $y(t)$ with the trapezoidal integration formula,

$$y(t) = \int_{t_0}^t \frac{d}{d\tau}y(\tau)d\tau + y(t_0) \quad (\text{B.9})$$

where the approximate solution is given by:

$$y(t) = \frac{1}{2} \left[\frac{d}{dt}y(t) + \frac{d}{dt}y(t_0) \right] (t - t_0) + y(t_0) \quad (\text{B.10})$$

then using $\frac{d}{dt}y(t)$ from Eqn. (B.8) with $t = nT$ and $t_0 = (n-1)T$, Eqn. (B.10) can be expressed as follows:

$$(2 + aT)y(n) - (2 - aT)y(n-1) = bT[x(n) + x(n-1)] \quad (\text{B.11})$$

Taking the z-transform of this difference equation and using the time shifting property of the z-transform, Eqn. (B.5) results in the z-domain system function:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b}{\frac{2}{T} \left(\frac{1-z^{-1}}{1+z^{-1}} \right) + a} \quad (\text{B.12})$$

Clearly, the mapping between the s-plane and the z-plane is:

$$s = \frac{2}{T} \left(\frac{1-z^{-1}}{1+z^{-1}} \right) \quad (\text{B.13})$$

This mapping is called bilinear transformation.

Although this transformation was developed using a first-order system, it holds, in general, for an N^{th} -order system (see Reference 14). By letting $s = \sigma + j\Omega$ and $z = re^{j\theta}$, it can be shown that the left-half plane in the s-domain is mapped inside the unit $r = 1$ circle in the z-domain under the bilinear transformation. More importantly, when $r = 1$ and $\sigma = 0$, the frequencies in the s-domain and the z-domain are related by:

$$\Omega = \frac{2}{T} \tan \frac{\theta}{2} \quad (\text{B.14})$$

or equivalently:

$$\theta = 2 \tan^{-1} \frac{\Omega T}{2} \quad (\text{B.15})$$

where: θ is the digital domain normalized frequency equal to $2\pi f/f_s$

Ω is the analog domain frequency used in the analysis of the previous section

On the $j\Omega$ axis or equivalently along the frequency axis, the scale has been changed nonlinearly. The gain and phase values depicted on the vertical axis of Figure B.1, Figure B.4, Figure B.6, and Figure B.8 remain exactly the same in the digital domain (or z -plane). The horizontal (frequency) axis is modified so that an infinite frequency in the analog domain maps to one-half of the sample frequency, $f_s/2$, in the digital domain; whereas, for frequencies much less than $f_s/2$, the mapping is approximately 1:1 with $\theta = \Omega$. In summary, the bilinear transformation is a one-to-one nonlinear mapping from the s -domain into the z -domain in which high frequencies ($\Omega > 2\pi f_s/4$) in the s -domain are compressed into a small interval in the z -domain. Therefore, the gain and phase expressions of the previous section can be directly transformed into the digital domain by simply substituting Eqn. (B.14) into the corresponding expressions. This will be done for each filter type in the following paragraphs.

First, it is appropriate to introduce the direct-form implementation of a digital filter by noting that, in general, if the bilinear transformation of Eqn. (B.13) is substituted into the transfer function, $H(s)$, of the previous section, the resulting $H(z)$ will have the following generalized form:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \quad (\text{B.16})$$

where the digital domain coefficients, a_i and b_i , are exactly related to the s -domain characteristics of the system such as the center frequency, bandwidth, etc. In the direct-form implementation, the a_i and b_i are used directly in the difference equation, which can be easily programmed on a high-speed DSP such as the DSP56001. The time-domain difference equation is derived from the z -domain transfer functions by applying the inverse z -transform in general and the inverse time shifting property in particular as follows:

$$\begin{aligned} Z^{-1}\{H(z)\} &= Z^{-1}\{Y(z)/X(z)\} \\ &= Z^{-1}\left\{\left[b_0 + b_1 z^{-1} + b_2 z^{-2}\right] / \left[1 + a_1 z^{-1} + a_2 z^{-2}\right]\right\} \end{aligned} \quad (\text{B.17})$$

so that:

$$Z^{-1}\left\{Y(z)\left[1 + a_1 z^{-1} + a_2 z^{-2}\right]\right\} = Z^{-1}\left\{X(z)\left[b_0 + b_1 z^{-1} + b_2 z^{-2}\right]\right\}$$

therefore, using the inverse time shifting property of Eqn. (B.5):

$$Z^{-1} \left\{ X(z) z^{-k} \right\} = \{x(n-k)\}$$

and

$$Z^{-1} \left\{ Y(z) z^{-k} \right\} = \{y(n-k)\}$$

Eqn. (B.17) becomes:

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + a_1 y(n-1) - a_2 y(n-2) \quad (\text{B.18})$$

Eqn. (B.18) can be directly implemented in software, where $x(n)$ is the sample input and $y(n)$ is the corresponding filtered digital output. When the filter output is calculated using Eqn. (B.18), $y(n)$ is calculated using the direct-form implementation of the digital filter.

There are other implementations which can be used for the same system (filter) transfer function, $H(z)$. The canonic-form implementation and the transpose-form implementation are discussed in subsequent sections. First, the directform implementation will be applied to the transfer function, $H(s)$, developed in **SECTION B.1 Introduction**.

B.2.1 Digital Lowpass Filter

Using the analog transfer function, $H(s)$, from Figure B.1, Eqn. (B.13) and Eqn. (B.14), the digital transfer function, $H(z)$, becomes that shown in Figure B.10, where the coefficients α , β , and γ are expressed in terms of the digital cutoff frequency, θ_c , and the damping factor, d . The value of the transfer function at $\theta = \theta_c$ in the digital domain is identical to the value of the s -domain transfer function at $\Omega = \Omega_c$:

$$H_z(e^{j\theta_c}) = H_s(j\Omega_c) \quad (\text{B.19})$$

As shown in Figure B.11, the digital gain and phase response calculated from the equations of Figure B.10 are similar to the analog plots shown in Figure B.2, except for the asymmetry introduced by the zero at $f_s/2$. That is, the frequency axis is modified so that a gain of zero at $f = \infty$ in the s -domain corresponds to a gain of zero at $f = f_s/2$ in the z -domain. The fact that the magnitude of the transfer functions, $H(s)$ and $H(z)$, is identical once the proper frequency transformation is made is very useful for understanding the digital filter and its relationship to the analog equivalent. This fact is also useful for purposes of scaling the gain since the maximum magnitude of $G_s(\Omega_M) = G_z(\theta_M)$, where Ω_M and θ_M are related by Eqn. (B.14). In other words, scaling analysis of the digital transfer function, $H(z)$, can be done in the s -domain (the algebra is often easier to manage). Scaling of the gain is an essential part of digital filter implementation since the region of numeric calculations on fixed-point DSPs such as the DSP56001 is usually restricted to a range of -1 to 1.

Using the formulas given in Figure B.10 with Eqn. (B.2) guarantees the behavior of the digital filter. Since the gain is scaled to unity at $f = 0$ (DC), the input data, $x(n)$, in Figure B.10 must be scaled down by a factor of $1/G_M$ from Eqn. (B.2) if the entire dynamic range of the digital network is to be utilized. The alternative procedure is automatic gain control to insure that $x(n)$ is smaller than $1/G_M$ before it arrives at the filter input. For $d \geq \sqrt{2}$, the input does not require scaling since the gain of the filter will never exceed unity.

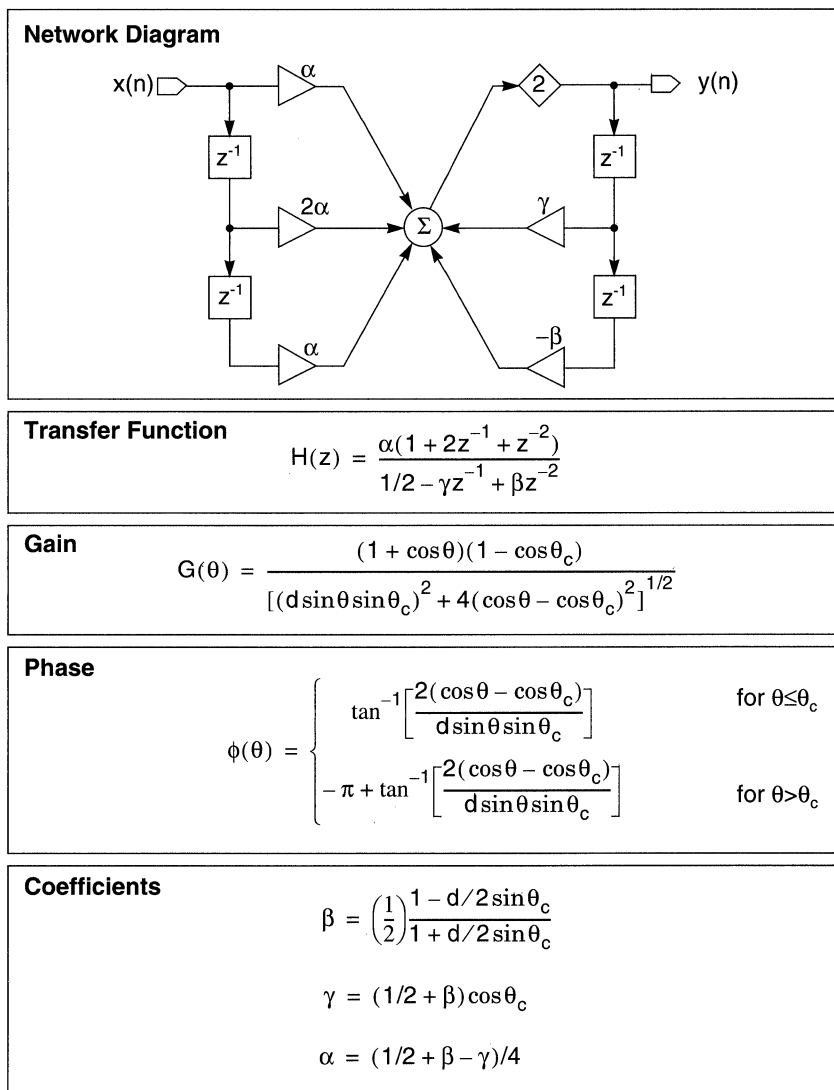


Fig. B.10 Direct-Form Implementation of Second-Order Lowpass IIR Filter and Analytical Formulas Relating Desired Response to Filter Coefficients

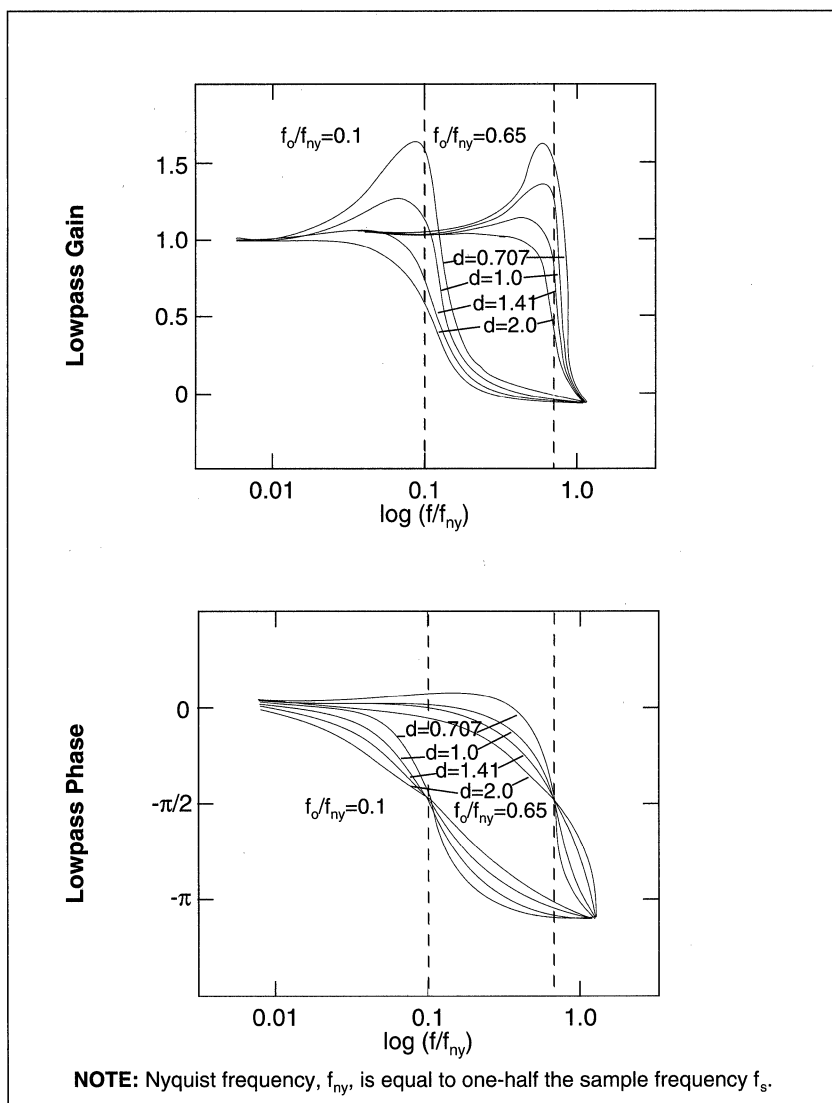


Fig. B.11 Gain and Phase Response of Second-Order Lowpass IIR Filter

The DSP56001 code to implement the second-order lowpass filter section is shown in Figure B.12. The address register modifiers are initially set to $M0 = 4$, $M4 = 1$, and $M5 = 1$ to allow use of the circular buffer or modulo addressing in this particular implementation (see Reference 8). Typically, this code would be an interrupt

Difference Equation

$$y(n) = 2\{\alpha[x(n) + 2x(n-1) + x(n-2)] + \gamma y(n-1) - \beta y(n-2)\}$$

Data Structures

X:(R0)	2 α	Y:(R4)	x(n-1)
	α		x(n-2)
	γ	Y:(R5)	y(n-1)
	- β		y(n-2)
	α		

DSP56001 Code

```

;Y1=x(n) (Input)
;X0=a
MPY   X0,Y1,A      X:(R0)+,X0  Y:(R4)+,Y0      ;A=ax(n)
MAC   X0,Y0,A      X:(R0)+,X0  Y:(R4),Y0      ;A=A+2ax(n-1)
MAC   X0,Y0,A      X:(R0)+,X0  Y:(R5)+,Y0      ;A=A+ax(n-2)
MAC   X0,Y0,A      X:(R0)+,X0  Y:(R5),Y0      ;A=A+gy(n-1)
MAC   X0,Y0,A      X:(R0)+,X0  Y1,Y:(R4)      ;A=A-by(n-2)
MOVE  A,X1         A,Y:(R5)    ;y(n-2)=2A (assumes scaling
                                ;mode is set).
                                ;X1 is Output.
```

Total Instruction Cycles

6 lcyc @ 20 MHz = 600ns

Fig. B.12 DSP56001 Code and Data Structures for Second-Order Direct-Form Implementation of a Lowpass IIR Filter

routine driven by the input data (A/D converter, for example) sample rate clock. The basic filter code and the interrupt overhead and data I/O moves for this second-order filter could be executed at a sample rate of nearly 1 MHz on the DSP56001.

B.2.2 Digital Highpass Filter

The highpass filter is nearly identical to the lowpass filter as shown by the formulas in Figure B.13. As with the analog case, the digital highpass filter is just the mirror image of the lowpass filter (see Figure B.14). The frequency transformation from high to low in the analog case is $\Omega \rightarrow 1/\Omega$; whereas, in the digital case, it is $\theta \rightarrow \pi/\theta$.

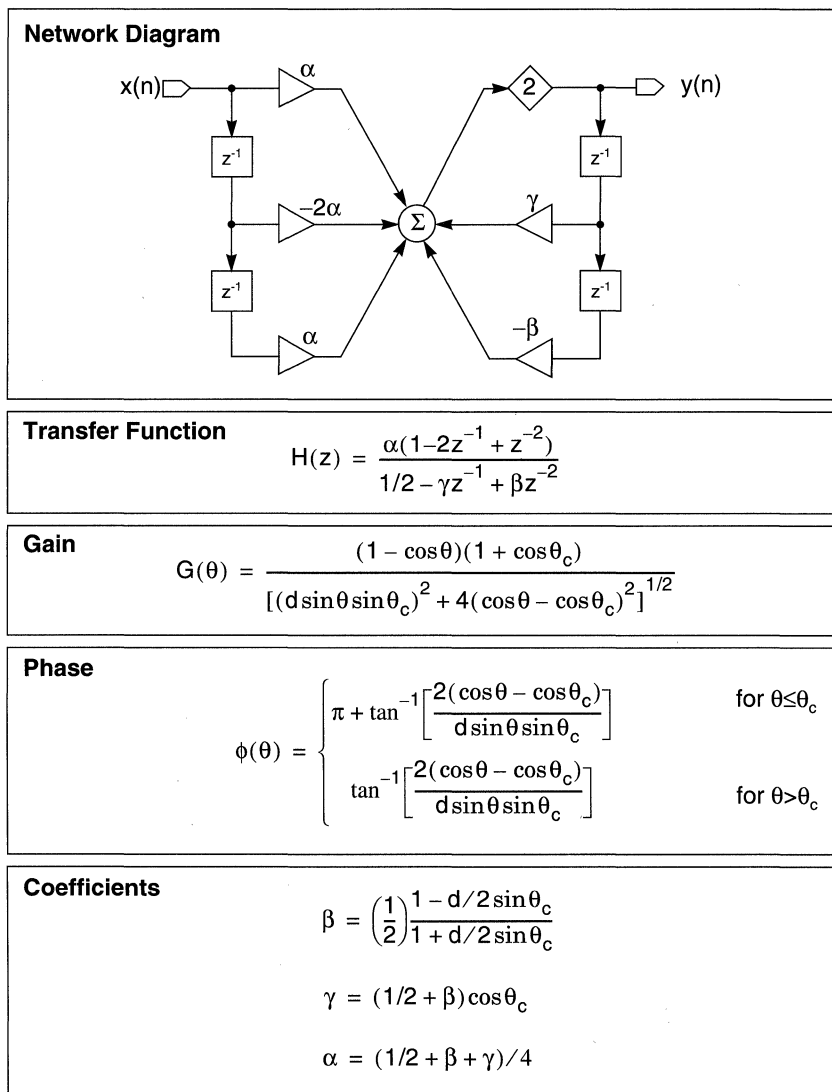


Fig. B.13 Direct-Form Implementation of Second-Order Highpass IIR Filter and Analytical Formulas Relating Desired Response to Filter Coefficients

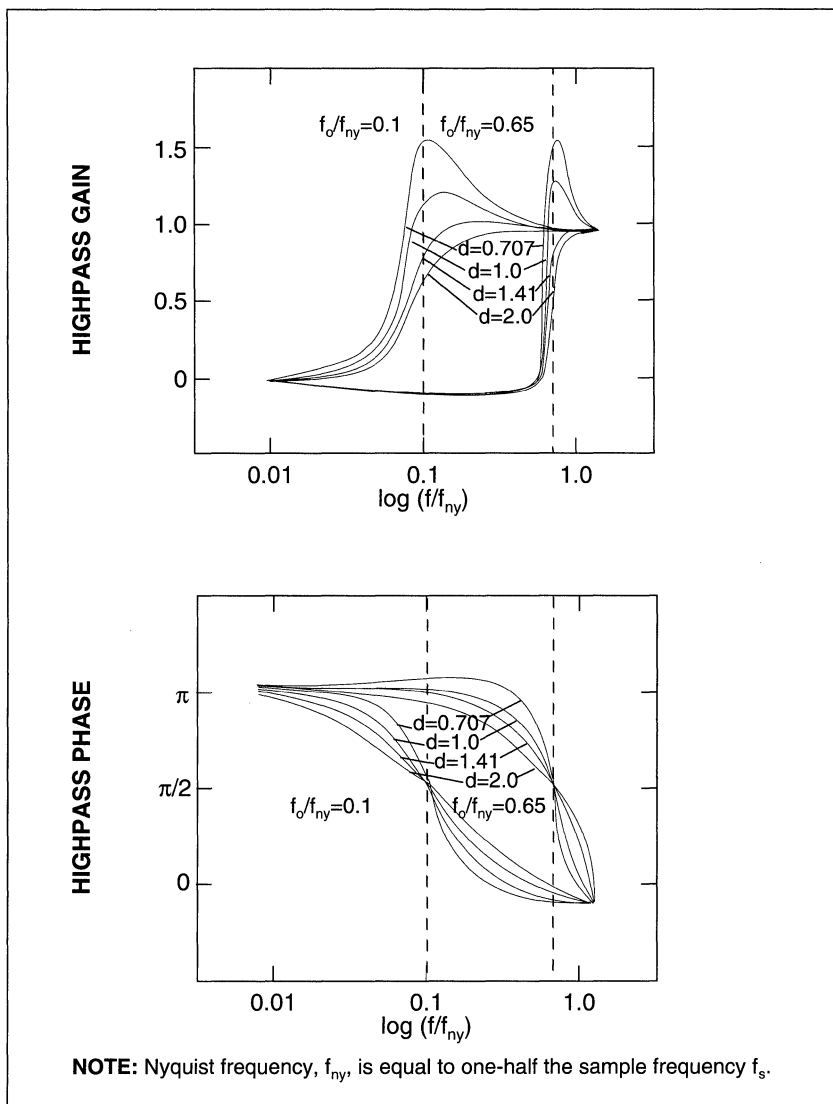


Fig. B.14 Gain and Phase Response of Second-Order Highpass IIR Filter

The DSP56001 code is shown in Figure B.15; as seen by comparison to the code shown in Figure B.12, the same instruction sequence is used. The only difference is the coefficient data, which is calculated by the formulas given in Figure B.13. The scaling mode is turned on so that a move from the A or B accumulator to the X or Y register or to memory results in an automatic multiply by two. The scaling mode is used not only in the code for the lowpass case but also in the code for the highpass, bandstop, and bandpass cases.

Difference Equation

$$y(n) = 2\{\alpha[x(n)-2x(n-1)+x(n-2)] + \gamma y(n-1) - \beta y(n-2)\}$$

Data Structures

X:(R0)	-2 α	Y:(R4)	x(n-1)
	α		x(n-2)
	γ	Y:(R5)	y(n-1)
	- β		y(n-2)
	α		

DSP56001 Code

```

;Y1=x(n) (Input)
;X0=a
MPY  X0,Y1,A    X:(R0)+,X0    Y:(R4)+,Y0    ;A=ax(n)
MAC  X0,Y0,A    X:(R0)+,X0    Y:(R4),Y0    ;A=A-2ax(n-1)
MAC  X0,Y0,A    X:(R0)+,X0    Y:(R5)+,Y0    ;A=A+ax(n-2)
MAC  X0,Y0,A    X:(R0)+,X0    Y:(R5),Y0    ;A=A+gy(n-1)
MAC  X0,Y0,A    X:(R0)+,X0    Y1,Y:(R4)    ;A=A-by(n-2)
MOVE A,X1      A,Y:(R5)      ;y(n)=2A (assumes scaling
                               ;mode is set).
                               ;X1 is Output.
```

Total Instruction Cycles

6 lcyc @ 20 MHz = 600ns

Fig. B.15 DSP56001 Code and Data Structures for Second-Order Direct-Form Implementation of a Highpass IIR Filter

B.2.3 Digital Bandstop Filter

The formulas and network diagram for the digital bandstop filter are presented in Figure B.16. The DSP56001 code from Figure B.17 is identical to that for the low-pass and highpass cases except for the coefficient data calculated from the equations of Figure B.16. Scaling of this filter is not a problem for the single-section case since

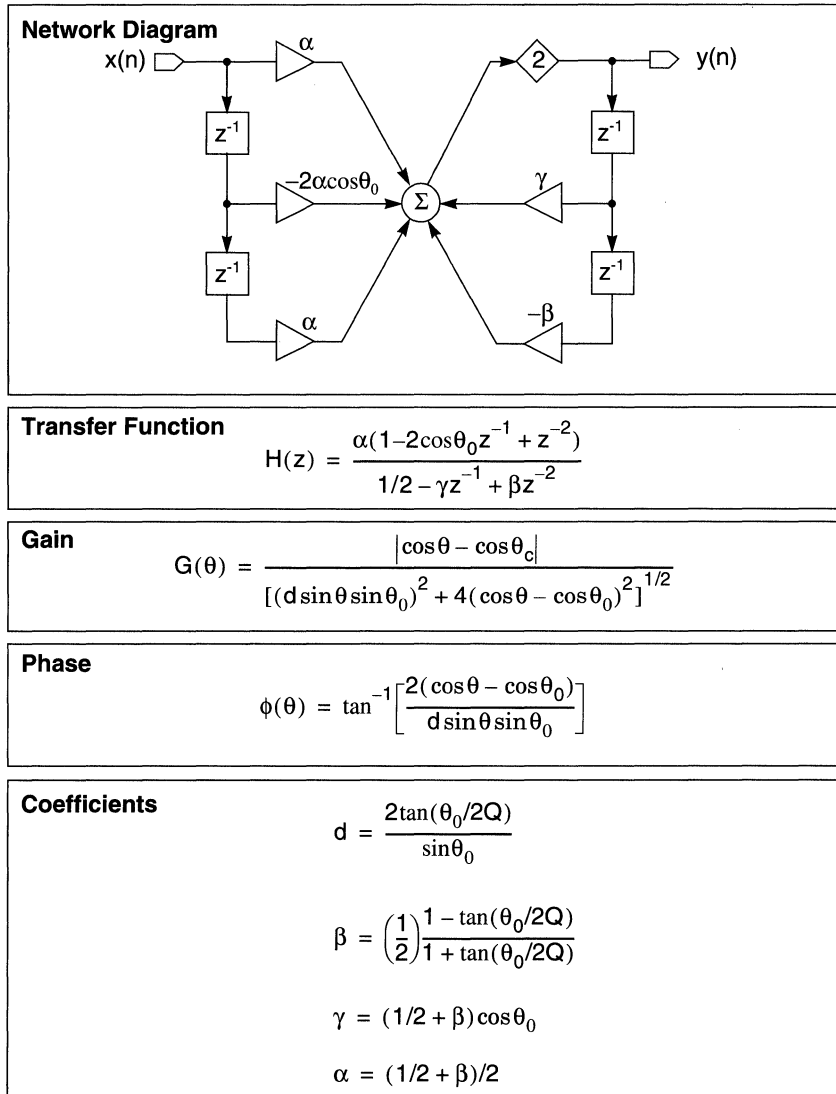


Fig. B.16 Direct-Form Implementation of Second-Order Bandstop IIR Filter and Analytical Formulas Relating Desired Response to Filter Coefficients

the gain from the equation in Figure B.16 never exceeds unity (as is true in the analog case as seen by the gain equation from Figure B.5). Figure B.18 is the calculated gain and phase of the digital filter, which should compare to the response curves of the equivalent analog filter plotted in Figure B.6.

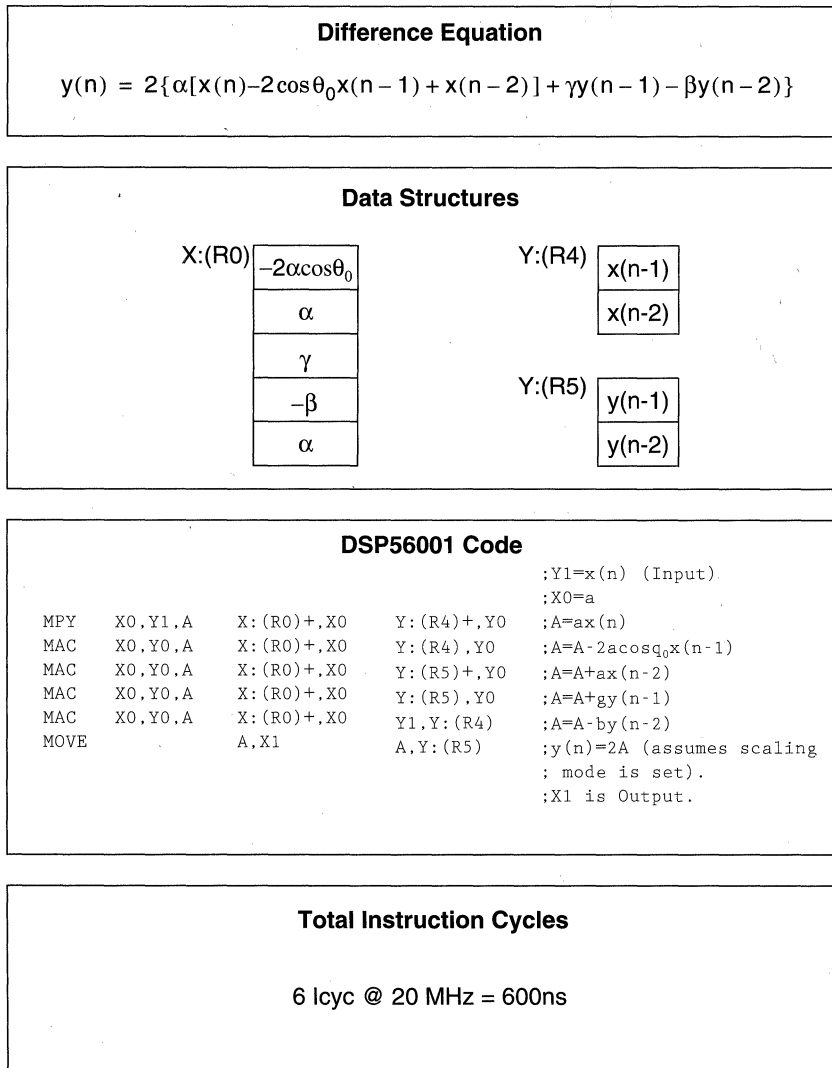


Fig. B.17 DSP56001 Code and Data Structures for Second-Order Direct-Form Implementation of a Bandstop IIR Filter

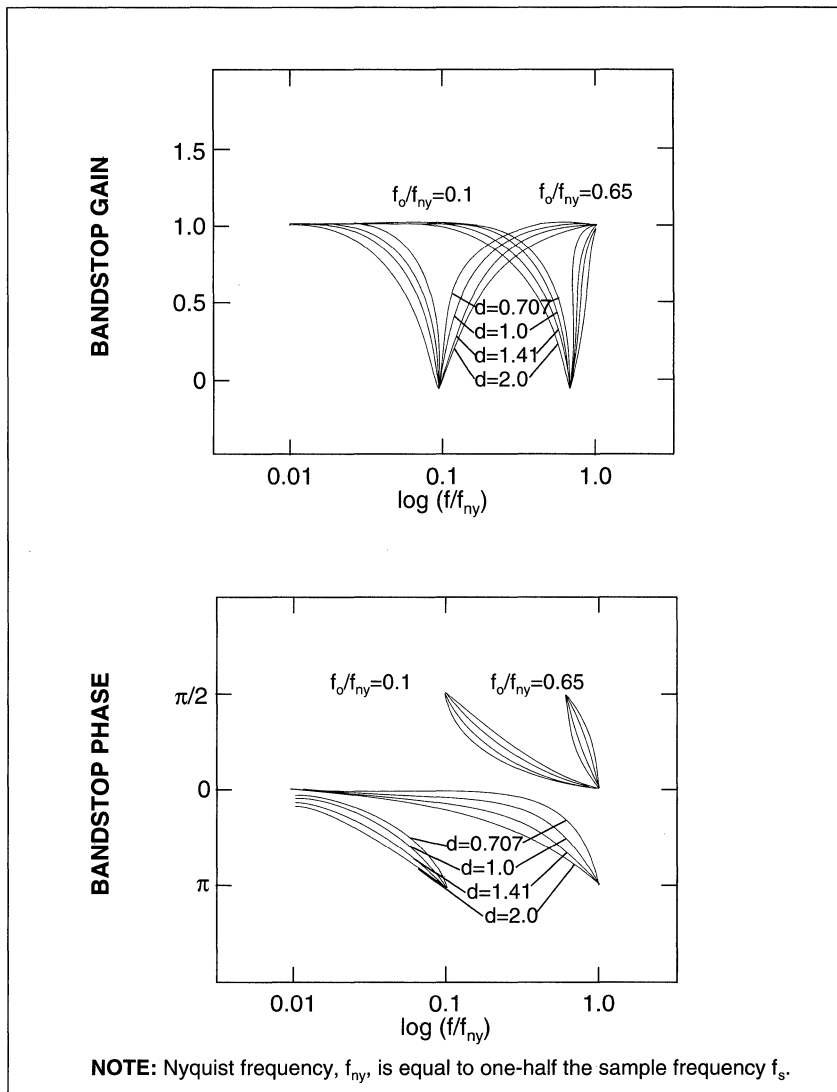


Fig. B.18 Gain and Phase Response of Second-Order Bandstop IIR Filter

B.2.4 Digital Bandpass Filter

Because there is one less coefficient in the bandpass network (see Figure B.19), one instruction can be saved in the DSP56001 code implementation shown in Figure B.20. Otherwise, the instructions are identical to those in the other three filter rou-

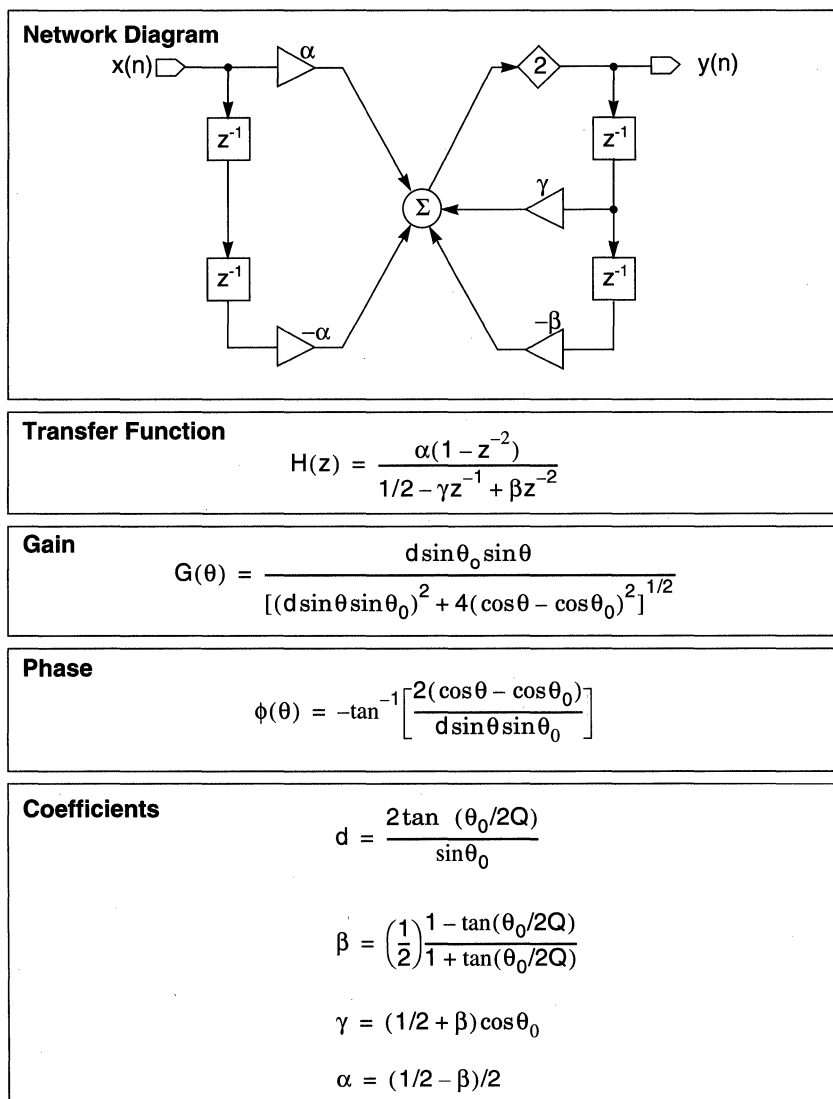


Fig. B.19 Direct-Form Implementation of Second-Order Bandpass IIR Filter and Analytical Formulas Relating Desired Response to Filter Coefficients

tines. Like the second-order bandstop network, the maximum response at the center frequency, θ_0 , is unity for any value of Q so that scaling need not be considered in the implementation of a single-section bandpass filter. This is true when the formulas for α , β , and γ (from Figure B.19) are used in the direct-form implementation in Figure B.20. Figure B.21 is the calculated gain and phase of the digital filter, which should compare to the response curves of the equivalent analog filter plotted in Figure B.8.

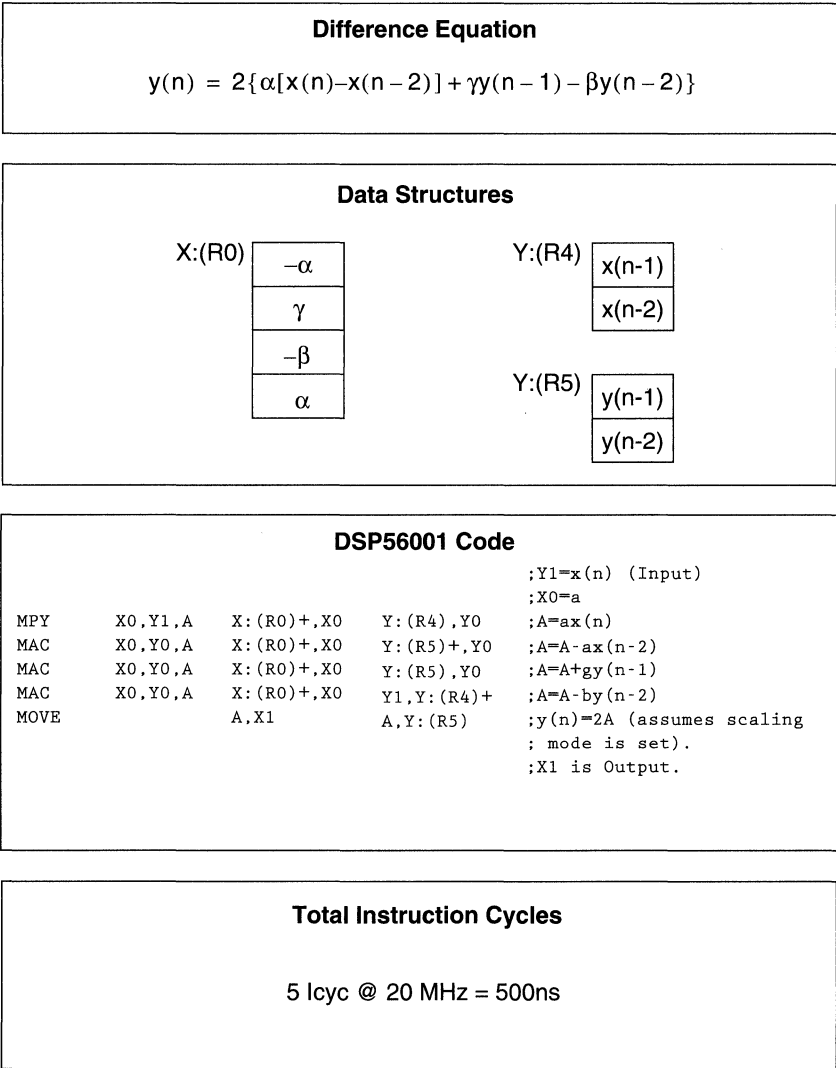


Fig. B.20 DSP56001 Code and Data Structures for Second-Order Direct-Form Implementation of a Bandpass IIR Filter

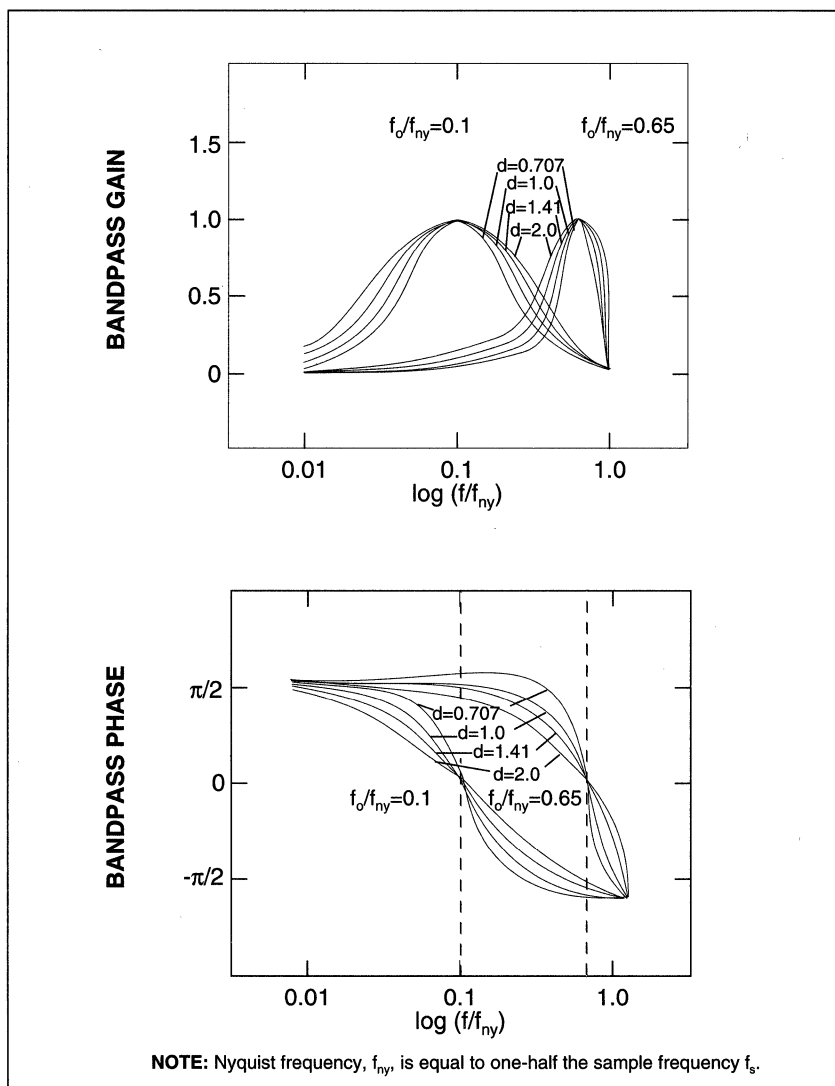


Fig. B.21 Gain and Phase Response of Second-Order Bandpass IIR Filter

B.2.5 Summary of Digital Coefficients

Figure B.22 gives a summary of the coefficient values for the four basic filter types. Note that the coefficient, β has the same form for all four filter types and that it can only assume values between 0 and $1/2$ for practical filters. β is bounded by $1/2$ because Q (or d) and θ_0 are not independent. For $Q \gg 1$, $\beta \rightarrow 1/2$; whereas, for $\theta_0 = f_s/4$ and $Q = 1/2$, $\beta \rightarrow 0$. These properties are independent of the form of implementation;

they are only dependent on the form of the transfer function. Alternate implementations (difference equations) will be described in the following sections.

Note that the Q described in Figure B.22 meets the traditional requirements (i.e., Q is the ratio of the bandwidth at the -3 dB points divided by the center frequency). The formula for β can be modified in the case of the bandpass or bandstop filter by replacing the damping coefficient, d , with the formula for Q . When the coefficients are described in this manner, a constant Q filter results. When the band-

Z-Domain Transfer Function

$$H(z) = \frac{\alpha(1 + \mu z^{-1} + \sigma z^{-2})}{1/2 - \gamma z^{-1} + \beta z^{-2}}$$

Difference Equation (Direct Form)

$$y(n) = 2\{\alpha[x(n) + \mu x(n-1) + \sigma x(n-2)] + \gamma y(n-1) - \beta y(n-2)\}$$

Coefficients

$$\beta = \frac{1}{2} \left[\frac{1 - 1/2 d \sin \theta_0}{1 + 1/2 d \sin \theta_0} \right] \quad d = \frac{2 \tan(\theta_0/2Q)}{\sin \theta_0} \quad \gamma = (1/2 + \beta) \cos \theta_0$$

where $0 < \beta < 1/2$ and $Q = \frac{\theta_0}{\Delta_0} = \frac{2\pi f_0/f_s}{2\pi(f_2 - f_1)/f_s} = \frac{f_0}{f_2 - f_1}$

where f_0 is the center frequency of the bandpass or bandstop filter, f_1 and f_2 are the half-power points (where gain is equal to $1/\sqrt{2}$), and f_s is the sample frequency. Note the f_0 is replaced with f_c in the lowpass and highpass cases.

Numerator Coefficients				
Type	α	μ	σ	Unity Gain at
Lowpass	$(1/2 + \beta - \gamma)/4$	2	1	$f = 0$
Highpass	$(1/2 + \beta + \gamma)/4$	-2	1	$f = f_s/2$
Bandpass	$(1/2 - \beta)/2$	0	-1	$f = f_0$
Bandstop	$(1/2 + \beta)/2$	$-2\cos\theta_0$	1	$f = 0$ and $f = f_s/2$

NOTE: $\theta_0 = 2\pi f_0/f_s$

Fig. B.22 Summary of Digital Coefficients for the Four Basic Filter Types

width is any function of center frequency, this relationship between d and Q makes it impossible to implement a bandpass or bandstop filter by replacing Q with the desired function of bandwidth and center frequency.

Figure B.23 shows the relationship between the pole of the second-order section and the center frequency. Note that the pole is on the real axis for $d > 2$, where d is also constrained by $d < 2/\sin \theta_0$.

Pole Equation of $H(z)$

$$Z_p = r \cos \theta_p + j r \sin \theta_p$$

$$= \gamma \pm j \sqrt{2\beta - \gamma^2}$$

For $d < 2$

$$= \frac{\cos \theta_0 \pm j \sin \theta_0 \sqrt{1 - (1/2d)^2}}{1 + \frac{1}{2} d \sin \theta_0}$$

where: $\beta = \frac{1}{2}(2 - d \sin \theta_0)/(2 + d \sin \theta_0)$ and $\gamma = (1/2 + \beta) \cos \theta_0$

Distance from Origin to Pole is $|Z_p| = \sqrt{2\beta}$

For $d > 2$

$$Z_p = \gamma - \sqrt{\gamma^2 - 2\beta}$$

$$= \frac{\cos \theta_0 - \sin \theta_0 \sqrt{(1/2d)^2 - 1}}{1 + \frac{1}{2} d \sin \theta_0}$$

where: $\theta_p = 0$. To satisfy requirement $0 < \beta < 1/2$ results in $\frac{1}{2} d \sin \theta_0 < 1$

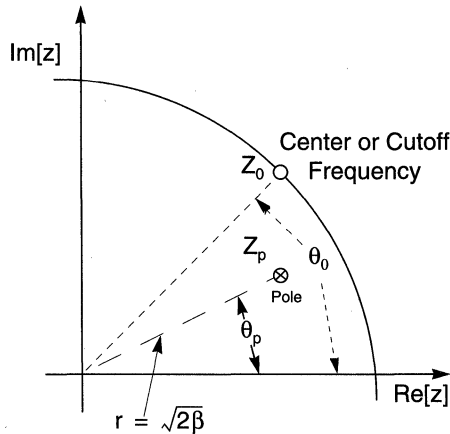


Fig. B.23 Pole Location and Analysis of Second-Order Section

B.3 SINGLE-SECTION CANONIC FORM (DIRECT FORM II)

“The canonic (direct form II) network has trade-offs that must be carefully understood and analyzed for the particular application.”

The single-section canonic form network is discussed in the following paragraphs.

B.3.1 The Canonic-Form Difference Equation

The direct-form difference equation, rewritten from Eqn. (B.18), is:

$$y(n) = \sum_{i=0}^2 b_i x(n-i) - \sum_{j=1}^2 a_j y(n-j) \quad (\text{B.20})$$

Eqn. (B.20) can be represented by the diagram of Figure B.24 (a). This diagram is the same as those shown in Figure B.10, Figure B.13, Figure B.16, and Figure B.19, except that the summations have been separated to highlight the correspondence with Eqn. (B.20). From this diagram, it is clear that the direct-form implementation requires four delay elements or, equivalently, four internal memory locations.

The diagram of Figure B.24 (b) represents the same transfer function implemented by Eqn. (B.20), but now the delay variable is $w(n)$. Comparison of this network with that of the direct-form network of Figure B.24 (a) shows that interchanging the order of the left and right halves does not change the overall system response (see Reference 11).

The delay elements can then be collapsed to produce the final canonic-form network shown in Figure B.24 (c). As a result, the memory requirements for the system are reduced to the minimum (two locations); therefore, this realization of the IIR filter is often referred to as the canonic form. The system difference equations for the canonic realization are:

$$y(n) = \sum_{i=0}^2 b_i (w(n-i)) \quad (\text{B.21a})$$

where:

$$w(n) = x(n) - \sum_{j=1}^2 a_j (w(n-j)) \quad (\text{B.21b})$$

To prove that Eqn. (B.21a) and Eqn. (B.21b) are equivalent to Eqn. (B.20), the following procedure can be used. First, combine Eqn. (B.21a) and Eqn. (B.21b):

$$y(n) = \sum_{i=0}^2 b_i \left[x(n-i) - \sum_{j=1}^2 a_j w(n-j-i) \right]$$

$$\begin{aligned}
 &= \sum_{i=0}^2 b_i x(n-i) - \sum_{j=1}^2 a_j \sum_{i=0}^2 b_i w(n-i-j) \\
 &= \sum_{i=0}^2 b_i x(n-i) - \sum_{j=1}^2 a_j y(n-j)
 \end{aligned} \tag{B.22}$$

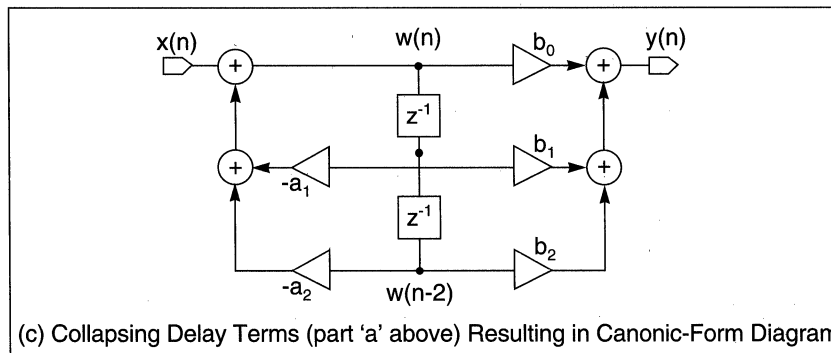
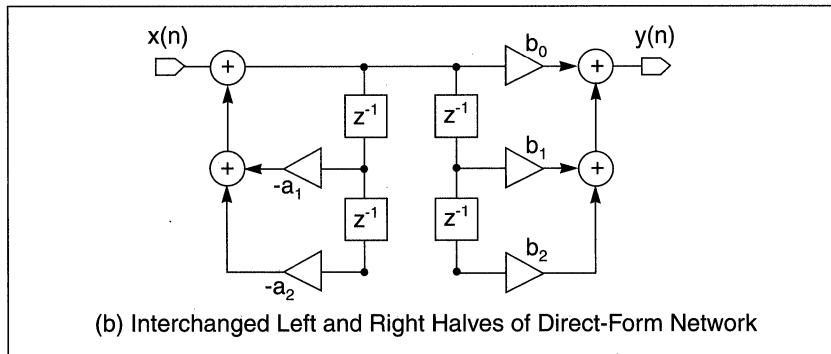
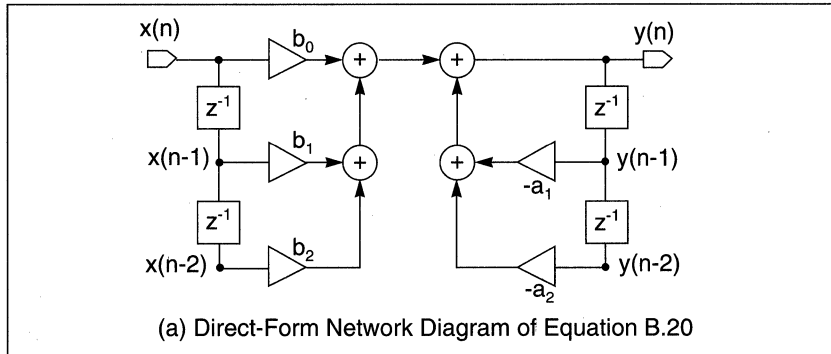


Fig. B.24 The Second-Order Canonic (Direct Form II) IIR Filter Network

The last step uses the definition for $y(n)$ from Eqn. (B.21a). The result is exactly equivalent to Eqn. (B.20), the direct-form difference equation. To utilize the scaling mode on the DSP56001, it is advantageous to write Eqn. (B.21a) and Eqn. (B.21b) as follows:

$$y(n) = 2 \left\{ \frac{1}{2} w(n) + \frac{\mu}{2} w(n-1) + \frac{\alpha}{2} w(n-2) \right\} \quad (\text{B.23a})$$

and

$$w(n) = 2 \{ \alpha x(n) + \gamma w(n-1) - \beta w(n-2) \} \quad (\text{B.23b})$$

where: the coefficients have been substituted for the a_j and b_j

From these equations, it can be seen that both $y(n)$ and $w(n)$ depend on a sum of products and therefore are the output of an accumulator. The accumulators on the DSP56001 have eight extension bits; thus, the sum can exceed unity by 255 without overflowing. However, when the contents of a 56-bit accumulator are transferred to a 24-bit register or memory location (i.e., delay element), an overflow error may occur. The digital filter designer must insure that only values less than unity are stored. Of course, even if memory had the precision of the accumulators, overflow can still occur if the accumulator itself experiences overflow. However, intermediate sums are allowed to exceed the capacity of the accumulator if the final result can be represented in the accumulator (a result of the circular nature of twos-complement arithmetic).

To insure that overflow does not occur, it is necessary to calculate the gain at the internal nodes (accumulator output) of a filter network. Although canonic realization is susceptible to overflow, it is advantageous because of the minimum storage requirements and implementation in fewer instruction cycles.

B.3.2 Analysis of Internal Node Gain

Calculating the gain at any internal node is no different than calculating the total network gain. In Figure B.25, the transfer function of node $w(n)$ is $H_w(z)$. $H_w(z)$ represents the transfer function from the input node, $x(n)$, to the internal node, $w(n)$. The gain, $G_w(\theta)$, at $w(n)$ is found in the standard manner by evaluating the magnitude of $H_w(z)$ as shown in Figure B.26. Note that the flow diagram of Figure B.25 uses the automatic scaling mode (multiply by a factor of two when transferring data from the accumulator to memory) so that the coefficients are by definition less than one. The peak gain, g_o , of $G_w(\theta)$ is found by taking the derivative of $G_w(\theta)$ with respect to θ and setting the result equal to zero. A simple expression for g_o is derived as follows:

$$g_o = \frac{\alpha}{\left(\frac{1}{2} - \beta\right) \sin \theta_p} \text{ for } \left[\frac{\left(\frac{1}{2} + \beta\right)^2}{2\beta} \right] \cos \theta_o \leq 1 \quad (\text{B.24})$$

where: θ_p is the angle (see Figure B.23) to the pole of the filter

Since this result does not depend on the numerator of the filter transfer function, $H(z)$, the result is valid for all four basic filter types. However, as shown by the example given for a bandpass network in Figure B.26, g_o may exceed unity by a large amount, especially for filters having poles at frequencies much less than $f_s/2$ where $\sin \theta_p \ll 1$. To compensate, the input must be scaled down by an amount equal to $1/g_o$ or guaranteed not to exceed $1/g_o$ before arriving at the filter. This scaling aspect of the canonic-form network is a disadvantage, but this network has the advantage of being

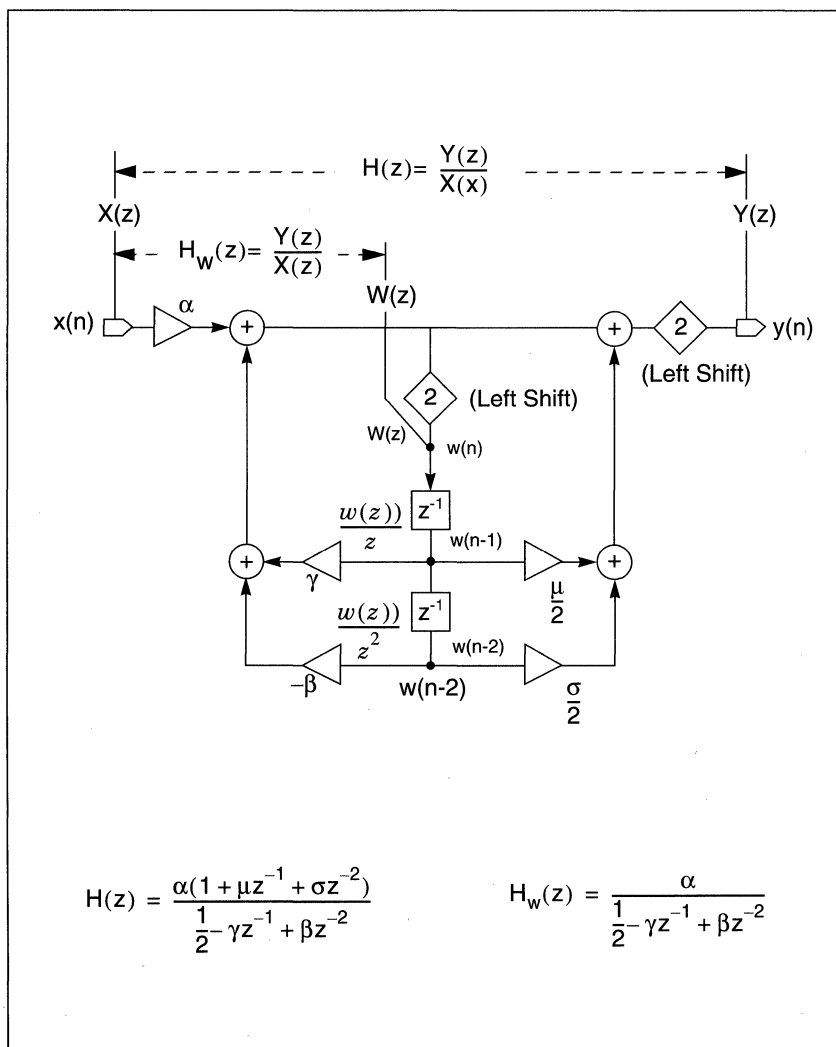


Fig. B.25 Internal Node Transfer Function, $H_w(z)$, of Canonic (Direct Form II) Network

implemented in one less instruction than the other filter realizations for the lowpass, highpass, and bandstop filters; of course, less memory is required since only two intermediate variables are stored.

In general, the behavior of systems at internal nodes can be unexpected. For instance, it is interesting to note that the frequency at which the gain at the internal node of the canonic IIR filter section peaks is not the same as the frequency at which

Gain at $w(n)$ is:

$$H_w(z) = \frac{W(z)}{X(z)} = \frac{\alpha}{\frac{1}{2} - \gamma z^{-1} + \beta z^{-2}}$$

$$\begin{aligned} G_w(\theta) &= \sqrt{H_w(e^{j\theta}) H_w(e^{-j\theta})} \\ &= \frac{\alpha}{\sqrt{\left(\frac{1}{2} - \beta\right)^2 \sin^2 \theta + \left(\frac{1}{2} + \beta\right)^2 (\cos \theta - \cos \theta_0)^2}} \end{aligned}$$

where: $\gamma = \left(\frac{1}{2} + \beta\right) \cos \theta_0$ has been used.

Peak Gain is: $\frac{d}{d\theta} G_w(\theta) \big|_{\theta = \theta_m} = 0$

Frequency of peak gain is: $\cos \theta_m = \xi \cos \theta_0$ for $\xi \cos \theta_0 \leq 1$
 $= 1$ otherwise

where: $\xi = \frac{\left(\frac{1}{2} + \beta\right)^2}{2\beta}$

$$\begin{aligned} g_0 \equiv G_w(\theta_m) &= \frac{\alpha}{\left(\frac{1}{2} - \beta\right) \sqrt{1 - \frac{\gamma^2}{2\beta}}} \quad \text{for } \xi \cos \theta_0 \leq 1 \\ &= \frac{\alpha}{\left(\frac{1}{2} + \beta\right) (1 - \cos \theta_0)} \quad \text{otherwise} \end{aligned}$$

EXAMPLE: Maximum Internal Node Gain for Bandpass Filter

$$g_0 = \frac{1}{2 \sin \theta_p} \quad \text{for } \xi \cos \theta_0 \leq 1$$

where $\gamma^2 = 2\beta^2 \cos^2 \theta_p$ has been used and $\alpha = (1/2 - \beta)/2$ for a bandpass filter. If $\sin \theta_p > 1/2$, then $g_0 < 1$; otherwise, an overflow (i.e., $g_0 > 1$) may occur at the internal node, $w(n)$, unless the input is scaled down by $1/g_0$.

Fig. B.26 Internal Node Gain Analysis of Second-Order Canonic Form

the gain of the filter peaks as given by the poles of the filter. From Figure B.13, this behavior is expressed by:

$$\frac{\sqrt{2\beta}}{\frac{1}{2} + \beta} = \frac{\cos \theta_o}{\cos \theta_p} \quad (\text{B.25})$$

The canonic (direct form II) network has trade-offs that must be carefully understood and analyzed for the particular application.

B.3.3 Implementation on the DSP56001

Figure B.27 shows the DSP56001 code and data structures for implementation of the single-section canonic-form network. Note that the modifier register M4 is set equal to 4 to allow circular operation for addressing coefficient data. M0 is set to FFFF to turn off circular addressing for $w(n-1)$ and $w(n-2)$.

B.4 SINGLE-SECTION TRANSPOSE FORM

"The modifier register M4 is initially set to a value of 4 so that a circular buffer can be conveniently used to address the coefficient data."

A third realization of IIR filters is the transpose form (direct form I) shown in Figure B.28. This network implementation can be derived directly from the direct-form difference equation (see Figure B.28) or by taking the transpose of the canonic network (see References 10 and 11). The transpose realization is characterized by three accumulator operations. One reason for the popularity of this realization is that, like the canonic realization, it only requires two memory locations; however, unlike the canonic realization, it is much less prone to overflow at internal nodes. The disadvantage is that this realization requires more instructions to implement.

B.4.1 Gain Evaluation of Internal Nodes

Using the same techniques used to calculate $H_w(z)$ for the canonic realization, $H_u(z)$ and $H_v(z)$ are found as shown in Figure B.29 and Figure B.30. The resulting expressions, unlike the canonic-form results, depend on the numerator of the transfer function; thus, the internal gains, $G_u(\theta)$ and $G_v(\theta)$, have different forms for the different filter types. In the case of the bandpass filter, these results simplify significantly so that a closed-form expression for the maximum gain at the internal nodes can be derived by calculating the maxima of the gain functions. For the bandpass and bandstop networks, the maximum of $G_u(\theta)$ and $G_v(\theta)$ is $g_m = \beta + 1/2$. Since, $\beta < 1/2$, then $g_m < 1$ so that no overflow occurs at these nodes in the bandpass or bandstop case.

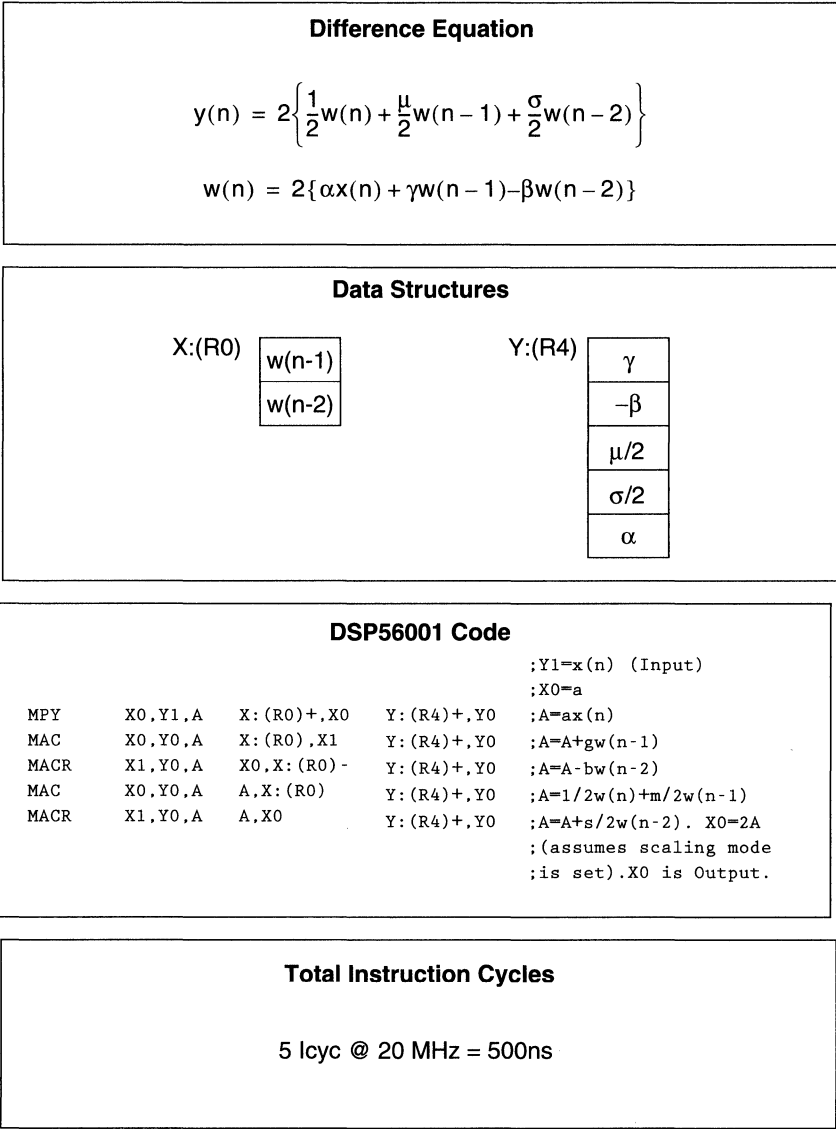
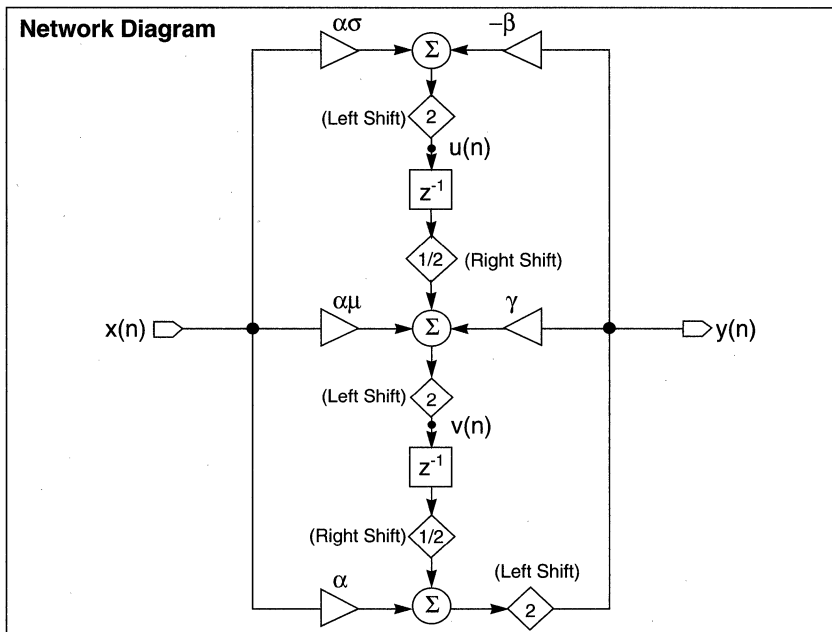


Fig. B.27 Internal Node Gain Analysis of Second-Order Canonic Form



Direct-Form Difference Equation

$$\begin{aligned}
 y(n) &= 2\{\alpha[x(n) + \mu x(n-1) + \sigma x(n-2)] + \gamma y(n-1) - \beta y(n-2)\} \\
 &= 2\{\alpha x(n) + \alpha \mu x(n-1) + \gamma y(n-1) + [\alpha \sigma x(n-2) - \beta y(n-2)]\} \\
 &= 2\left\{\alpha x(n) + [\alpha \mu x(n-1) + \gamma y(n-1) + \frac{1}{2}u(n-2)]\right\} \\
 &= 2\left\{\alpha x(n) + \frac{1}{2}v(n-1)\right\}
 \end{aligned}$$

Transpose-Form Difference Equation

$$\begin{aligned}
 y(n) &= 2\left\{\alpha x(n) + \frac{1}{2}v(n-1)\right\} \\
 \text{where } v(n) &= 2\left\{\alpha \mu x(n) + \gamma y(n) + \frac{1}{2}u(n-1)\right\} \\
 \text{and } u(n) &= 2\{\alpha \sigma x(n) - \beta y(n)\}
 \end{aligned}$$

Fig. B.28 Network Diagram for Transpose-Form (Direct Form I) Implementation of a Single-Section Second-Order Filter

GAIN AT INTERNAL NODE, $u(n)$

Transfer function of $u(n)$ is $U(z) = 2[\alpha\sigma X(z) - \beta Y(z)]$
 $= H_u(z)X(z)$

where $H_u(z) = 2[\alpha\sigma - \beta H(z)]$

$$= 2 \left[\frac{\alpha\sigma(1/2 - \gamma z^{-1} + \beta z^{-2}) - \alpha\beta(1 + \mu z^{-1} + \sigma z^{-2})}{1/2 - \gamma z^{-1} + \beta z^{-2}} \right]$$

$$= 2 \left[\frac{2\alpha(A - Bz^{-1})}{1/2 - \gamma z^{-1} + \beta z^{-2}} \right]$$

where $H_u(z)$ is the transfer function from input node, $x(n)$, to internal mode, $u(n)$, and $A = \sigma/2 - \beta$ and $B = \sigma\gamma + \mu\beta$.

Gain of $u(n)$ is $G_u(\theta) = |H_u(z)|_{z=e^{j\theta}}$

$$= \frac{2\alpha\sqrt{A^2 + B^2 - 2AB\cos\theta}}{\sqrt{(1/2 - \beta)^2 \sin^2\theta + (1/2 + \beta)^2 (\cos\theta - \cos\theta_0)^2}}$$

BANDPASS EXAMPLE

Bandpass Coefficients are $\sigma = -1$ $\mu = 0$ $\alpha = (1/2 - \beta)/2$

so that $A = -(1/2 + \beta)$

$B = -\gamma$

$= -(1/2 + \beta)\cos\theta_0$

$$\text{and } G_u(\theta) = \frac{(1/2 - \beta)(1/2 + \beta)\sqrt{\sin^2\theta + (\cos\theta - \cos\theta_0)^2}}{\sqrt{(1/2 - \beta)^2 \sin^2\theta + (1/2 + \beta)^2 (\cos\theta - \cos\theta_0)^2}}$$

Peak Gain is $g_m = G_u(\theta_m)$

which is found by evaluating $\frac{d}{d\theta} G_u(\theta)|_{\theta=\theta_m} = 0$

after evaluating derivative, $\theta_m = \theta_0$

$$\text{so that } g_m = \frac{(1/2 - \beta)(1/2 + \beta)\sin\theta_0}{(1/2 - \beta\sin\theta_0)} = 1/2 + \beta < 1$$

Fig. B.29 Gain Evaluation at First Internal Node, $u(n)$, of Transpose Network

GAIN AT INTERNAL NODE, $v(n)$

Transfer function of $v(n)$ is $V(z) = [Y(z) - 2\alpha X(z)]/z^{-1}$
 $= H_V(z)X(z)$

where $Y(z) = 2[\frac{1}{2}V(z)z^{-1} + \alpha X(z)]$ $H_V(z) = \frac{H(z) - 2\alpha}{z^{-1}}$

and $= \frac{\alpha[(1 + \mu z^{-1} + \sigma z^{-2}) - 2(1/2 + \gamma z^{-1} + \beta z^{-2})]}{z^{-1}(1/2 - \gamma z^{-1} + \beta z^{-2})}$

$= \frac{2\alpha(C + Az^{-1})}{1/2 - \gamma z^{-1} + \beta z^{-2}}$

where $H_V(z)$ is the transfer function from input node, $x(n)$, to internal mode, $v(n)$, and $A = \sigma/2 - \beta$ and $C = \mu/2 + \gamma$.

Gain of $v(n)$ is $G_V(\theta) = |H_V(z)|_{z=e^{j\theta}}$

$$= \frac{2\alpha\sqrt{A^2 + C^2 + 2AC\cos\theta}}{\sqrt{(1/2 - \beta)^2 \sin^2\theta + (1/2 + \beta)^2 (\cos\theta - \cos\theta_0)^2}}$$

BANDPASS EXAMPLE

Bandpass Coefficients are $\sigma = -1$ $\mu = 0$ $\alpha = (1/2 - \beta)/2$

so that $A = -(1/2 + \beta)$

$C = \gamma$
 $= (1/2 + \beta)\cos\theta_0$

and $G_V(\theta) = \frac{(1/2 - \beta)(1/2 + \beta)\sqrt{\sin^2\theta + (\cos\theta - \cos\theta_0)^2}}{\sqrt{(1/2 - \beta)^2 \sin^2\theta + (1/2 + \beta)^2 (\cos\theta - \cos\theta_0)^2}}$

Peak Gain is $g_m = G_V(\theta_m)$

which is found by evaluating $\frac{d}{d\theta} G_V(\theta)|_{\theta=\theta_m} = 0$

after evaluating derivative, $\theta_m = \theta_0$

so that $g_m = \frac{(1/2 - \beta)(1/2 + \beta)\sin\theta_0}{(1/2 - \beta)\sin\theta_0} = 1/2 + \beta < 1$

Fig. B.30 Gain Evaluation at Second Internal Node, $v(n)$, of Transpose Network

Figure B.31 contains example plots of $G_u(\theta)$ and $G_v(\theta)$ for the second-order transpose-form lowpass filter with various values of cutoff frequency, θ_c , and damping factor, d . In most cases, $G_u(\theta)$ and $G_v(\theta)$ never exceed the maximum value of $G(\theta)$, so that, if the total gain does not exceed unity, the internal nodes will not exceed unity (i.e., no overflow).

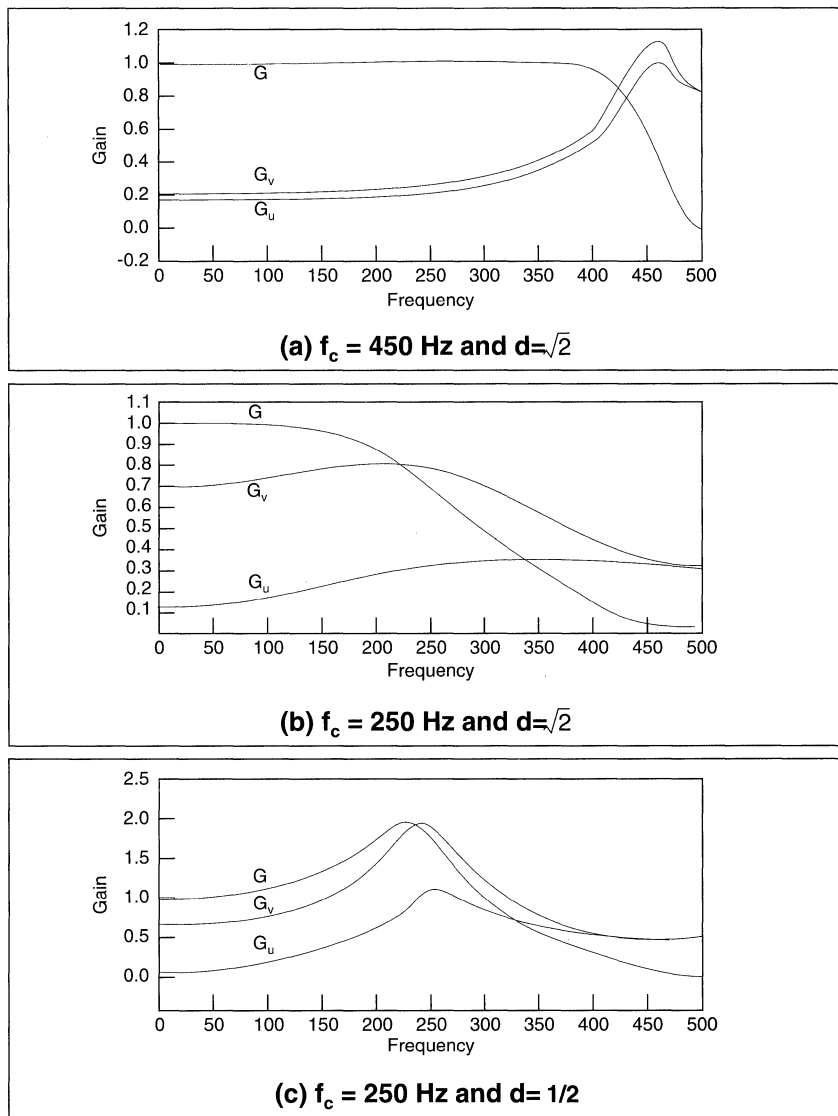


Fig. B.31 Total Gain and Gain at Internal Nodes of Lowpass Transpose Filter Network Figure B.28 for $f_s = 1000$ Hz

B.4.2 Implementation on the DSP56001

The DSP56001 code and data structures for a single-section second-order transpose-form network are shown in Figure B.32. Referring to the network diagram of Figure B.28, the diamond blocks enclosing $1/2$ are represented in the code by an accumulator shift

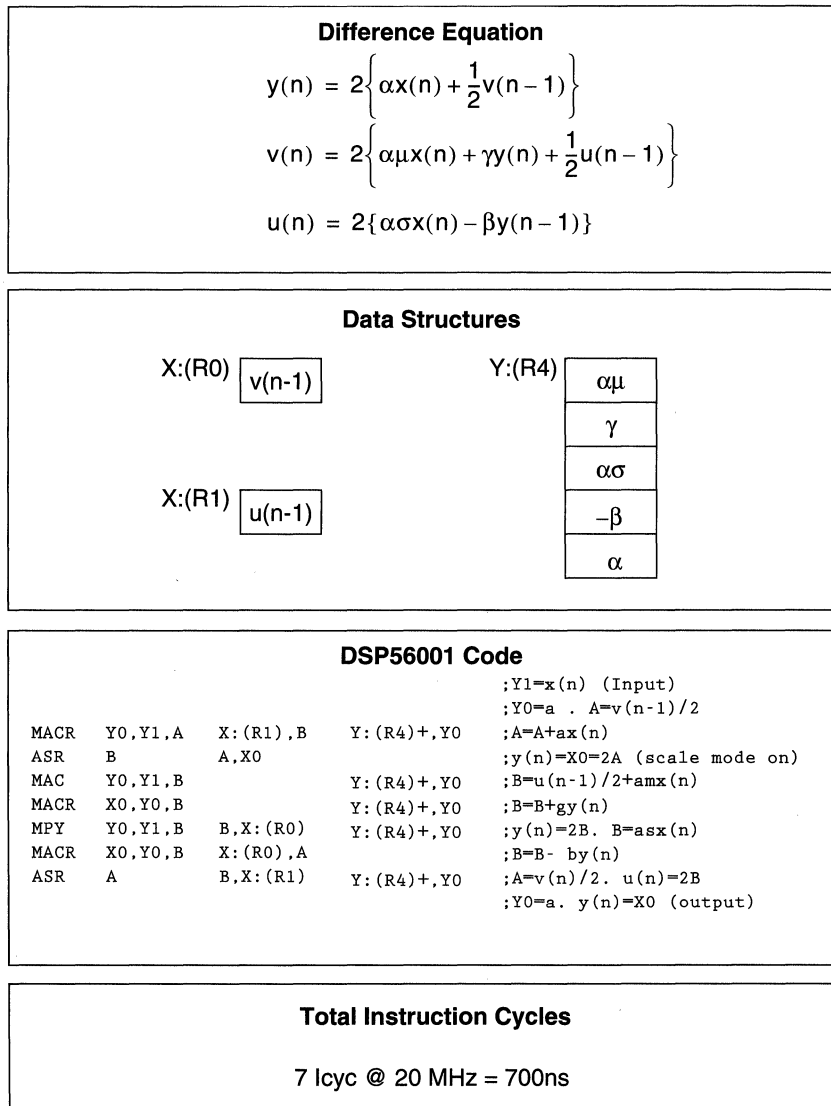


Fig. B.32 DSP56001 Code and Data Structures for Single-Section Second-Order Transpose Form

right (ASR) instruction. The 2 that is enclosed by a diamond block can be implemented by the automatic scaling mode (equivalent to a left shift of the accumulator) feature of the DSP56001. The modifier register M4 is initially set to a value of 4 so that a circular buffer can be conveniently used to address the coefficient data. Note that this network requires more instructions than either of the previous two network forms.

B.5 CASCADED DIRECT FORM

“... the cascaded direct-form network becomes canonic as the filter order N increases.”

By placing any of the direct-form second-order filter networks from Figure B.10, Figure B.13, Figure B.16, and Figure B.19 in series (i.e., connect the $y(n)$ of one to the $x(n)$ of the next), a cascaded filter is created. The resulting order N of the network is two times the number of second-order sections. An odd-order network can be made simply by adding one first-order section in the chain. In general, to achieve a particular response, the filter parameters associated with each second-order section are different, since generating a predefined total response requires that each section has a different response. This fact becomes more obvious when discussing the special case of Butterworth lowpass filters.

B.5.1 Butterworth Lowpass Filter

The Butterworth filter response is maximally flat in the passband at the expense of phase linearity and steepness of attenuation slope in the transition band. For low-pass or highpass cascaded second-order sections, all sections have the same center frequency (not the case for bandpass filters). For this reason, it is easy to design since all that remains to be determined are the damping factors, d_k , of each individual k th section. The damping coefficients, d_k , are calculated from a simple formula for any order N of response.

The s-domain transfer function for the N^{th} -order lowpass Butterworth filter is:

$$\begin{aligned}
 H(s) &= \left(\frac{1}{(s/\Omega_c)^2 + d_1(s/\Omega_c) + 1} \right) \left(\frac{1}{(s/\Omega_c)^2 + d_{2(1)}(s/\Omega_c) + 1} \right) \dots \\
 &= \prod_{k=1}^{N/2} \frac{1}{(s/\Omega_c)^2 + d_k(s/\Omega_c) + 1}
 \end{aligned} \tag{B.26}$$

where: d_k is the k^{th} damping coefficient,

$s = j\Omega$,

Ω_c is the common cutoff frequency (see Reference 14).

Only filter orders of even N will be considered in this discussion to minimize the complexity of mathematical results. The analysis can be extended to include odd val-

ues of N by inserting an additional term of $[(s/\Omega_c) + 1]^{-1}$ in Eqn. (B.26). Eqn. (B.26) can be generalized if Ω_c of each second-order section is an arbitrary value (corresponding to a different cutoff frequency, Ω_k , of each section). However, since this discussion is limited to Butterworth polynomials, Eqn. (B.26) will serve as the basis of all following derivations. A filter of order N has $N/2$ second-order sections. The second-order section of a Butterworth filter can be derived from the simple RCL network of Figure B.1. However, the Butterworth damping factors are predetermined values, which can be shown to yield a maximally flat passband response (see Reference 14). The Butterworth damping coefficients are given by the following equation:

$$d_k = 2 \sin \frac{(2k-1)\pi}{2N} \quad (\text{B.27})$$

Eqn. (B.27) is the characteristic equation that determines a Butterworth filter response. Note that for a single-section ($k = 1$) second-order ($N = 2$) lowpass filter, $d_k = 2 \sin(\pi/4) = \sqrt{2}$ as expected for a maximally flat response. For a fourth-order filter with two second-order sections, $d_1 = 2 \sin(\pi/8)$, where $k = 1$ and $N = 4$, and $d_2 = 2 \sin(3\pi/8)$, where $k = 2$ and $N = 4$.

Eqn. (B.26) represents an all-pole response (the only zeros are at plus and minus infinity in the analog s -domain). The poles of a second-order section are the roots of the quadratic denominator as given by:

$$p_{k1} = -d_k/2 - j \left(1 - d_k^2/4\right)^{1/2} \quad (\text{B.28a})$$

and

$$p_{k2} = d_k/2 + j \left(1 - d_k^2/4\right)^{1/2} \quad (\text{B.28b})$$

Using Eqn. (B.28a) and Eqn. (B.28b), Eqn. (B.26) becomes:

$$H(s) = \prod_{k=1}^{N/2} \frac{1}{[(s/\Omega_c) - p_{k1}][(s/\Omega_c) - p_{k2}]} \quad (\text{B.29})$$

where: $p_k = p_{k1}$ and

$p_k^* = p_{k2}$ (complex conjugate of p_k)

Eqn. (B.29) is useful in that the response of the system can be analyzed entirely by studying the poles of the polynomial. However, for purposes of transforming to the z -domain, Eqn. (B.26) can be used as previously shown in Figure B.10.

To examine the gain and phase response (physically measurable quantities) of the lowpass Butterworth filter, the transfer function, $H(s)$, of Eqn. (B.26) will be converted into a polar representation. The magnitude of $H(s)$ is the gain, $G(\Omega)$; the angle between the real and imaginary components of $H(s)$, $\phi(\Omega)$, is the arctangent of the phase shift introduced by the filter:

$$\begin{aligned}
G(\Omega) &= \sqrt{H(s)H^*(s)|_{s=j\Omega}} \\
&= \prod_{k=1}^{N/2} \frac{1}{\sqrt{\left[(\Omega/\Omega_c)^2 - 1\right]^2 + (d_k \Omega/\Omega_c)^2}}
\end{aligned} \tag{B.30}$$

$$\phi(\Omega) = \sum_{k=1}^{N/2} \tan^{-1} \frac{d_k \Omega/\Omega_c}{(\Omega/\Omega_c)^2 - 1} \tag{B.31}$$

Eqn. (B.30), and Eqn. (B.31) describe the response characteristics of an N^{th} -order lowpass Butterworth filter in the continuous frequency analog domain. Since the quantity of interest is usually $20 \log G(\Omega)$, Eqn. (B.30) can be transformed into a sum (over the second-order sections) of $20 \log (G_k)$, where G_k is the gain of the k^{th} section. Similarly, the total phase is just the sum of the phase contribution by each section from Eqn. (B.31).

The bilinear transformation is used to convert the continuous frequency domain transfer function into the digital domain representation, where θ , the normalized digital domain frequency equal to $2\pi f/f_s$, can be thought of as the ratio of frequency to sampling frequency scaled by 2π . Substituting s from Eqn. (B.13) and Ω_c from Eqn. (B.14) into the k^{th} section of Eqn. (B.26) yields the digital domain form of the Butterworth lowpass filter (for the k^{th} second-order section):

$$H_k(z) = \frac{\alpha_k(1 + 2z^{-1} + z^{-2})}{\frac{1}{2} - \gamma_k z^{-1} + \beta_k z^{-2}} \tag{B.32}$$

$$\alpha_k = \left[\tan^2(\theta_c/2) \right] / A_k(c) \tag{B.33a}$$

$$\beta_k = \left[1 - d_k \tan(\theta_c/2) + \tan^2(\theta_c/2) \right] / A_k(c) \tag{B.33b}$$

$$\gamma_k = 2 \left[1 - \tan^2(\theta_c/2) \right] / A_k(\theta_c) \tag{B.33c}$$

$$A_k(\theta_c) = 2 \left[1 + d_k \tan(\theta_c/2) + \tan^2(\theta_c/2) \right] \tag{B.33d}$$

Eqn. (B.33a)-Eqn. (B.33d) provides a complete description of the digital lowpass N^{th} -order Butterworth filter. Given θ_c and d_k , these formulas allow precise calculation of the digital coefficients, α_k , β_k , and γ_k , used to implement each k^{th} second-order section of the filter.

Eqn. (B.33a)-Eqn. (B.33d) can be further simplified into the following set of formulas:

$$\beta_k = \frac{1 - (d_k/2)\sin(\theta_c)}{2[1 + (d_k/2)\sin(\theta_c)]} \quad (\text{a}) \quad (\text{B.34a})$$

$$\gamma_k = (1/2 + \beta_k)\cos(\theta_c) \quad (\text{b}) \quad (\text{B.34b})$$

$$\alpha_k = (1/2 + \beta_k - \gamma_k)/4 \quad (\text{c}) \quad (\text{B.34c})$$

where: d_k is given by Eqn. (B.27)

θ_c is the digital domain cutoff frequency (actual operating cutoff frequency of the digital filter)

Figure B.33 shows an example of a sixth-order lowpass Butterworth filter (three second-order sections) in both the analog domain and digital domain. Note that the gain of the first section ($k = 1$) is greater than unity near the cutoff frequency but that the overall composite response never exceeds unity. This fact allows for easy implementation of the Butterworth filter in cascaded direct form (i.e., scaling of sections is not needed as long as the sections are implemented in the order of decreasing k). Overflow at the output of any section is then guaranteed not to occur (the gain of the filter never exceeds unity). Note that the digital response (see Figure B.33) is identical to the analog response but warped from the right along the frequency axis. Imagine the zero at plus infinity in the analog response mapping into the zero at $f_s/2$ in the digital case. Also note that, because of this mapping, the digital response falls off faster than the -12 dB/octave of the analog filter when the cutoff is near $f_s/2$.

The previous analysis is nearly identical to the case of the highpass filter except the coefficients (see Figure B.24) have slightly different values. Since the bandstop case is just the sum of a lowpass and highpass case, it can be analyzed by these techniques. The bandpass case, however, is more difficult and requires considerably more work (see Reference 14) because the center frequency of each section is now different and the formula for calculating these frequencies is not as simple as the formulas for the previous filter types. In addition, to complicate matters further, scaling between sections becomes more of a problem since the offset of the center of each section reduces the final center response, which must be compensated at some point in the filter network. For cases such as higher order Butterworth bandpass filter designs, commercially available filter design packages such as FDAS are useful. The use of FDAS is discussed in **SECTION B.6 Filter Design and Analysis System (FDAS)** and **SECTION B.7 Fir Filters**.

B.5.2 Cascaded Direct-Form Network

Figure B.34 shows the cascaded direct-form network and data structures for the DSP56001 code implementation of Figure B.35. By cascading network diagrams presented in **SECTION B.2 Second-Order Direct-Form IIR Digital Filter Sections**, the set of delays at the output of one section can be combined with the set of delays at the input of the next section, thus reducing the total number of delays by almost a factor of two. For this reason, the cascaded direct-form network becomes canonic as the

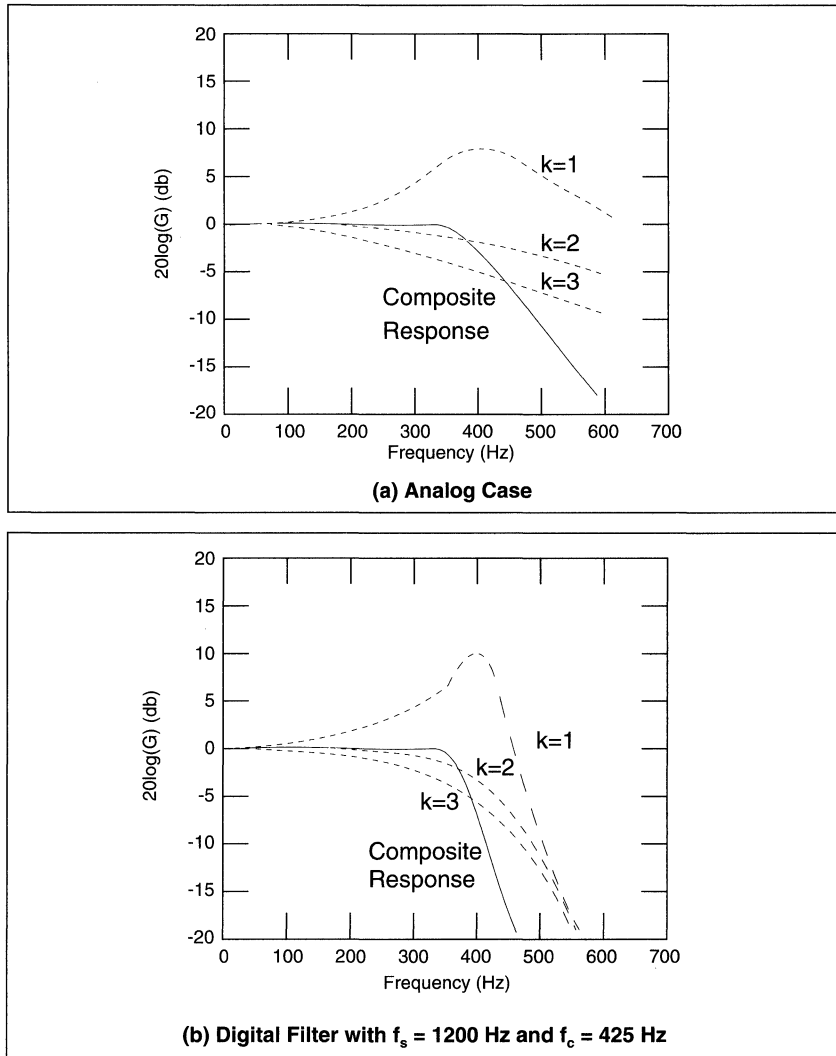


Fig. B.33 Composite Response of Cascaded Second-Order Sections for Lowpass Butterworth Filter (k^{th} Section Damping Factor According to Eqn. (B.27))

filter order N increases. The DSP56001 code (see Figure B.35) shows an example of reading data from a user-supplied memory-mapped analog-to-digital converter (ADC) and writing it to a memory-mapped digital-to-analog converter (DAC). The number of sections (nsec) in these examples is three; thus, the filter order is six. The total instruction time for this filter structure is $600 \text{ nsec} + 800 \text{ ns}$, including the data I/O moves (but excluding the interrupt overhead).

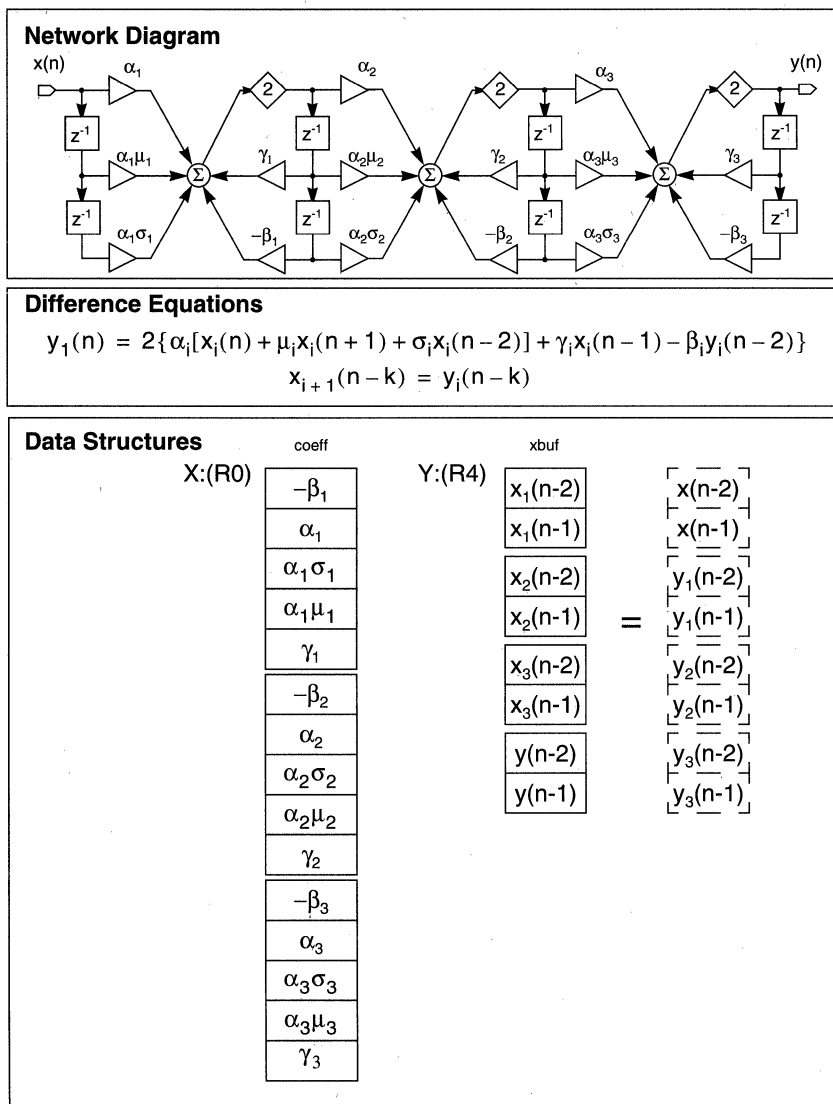


Fig. B.34 Network Diagram for Cascaded Direct-Form Filter and Data Structures Used in Code Implementation (Three-Section Example)

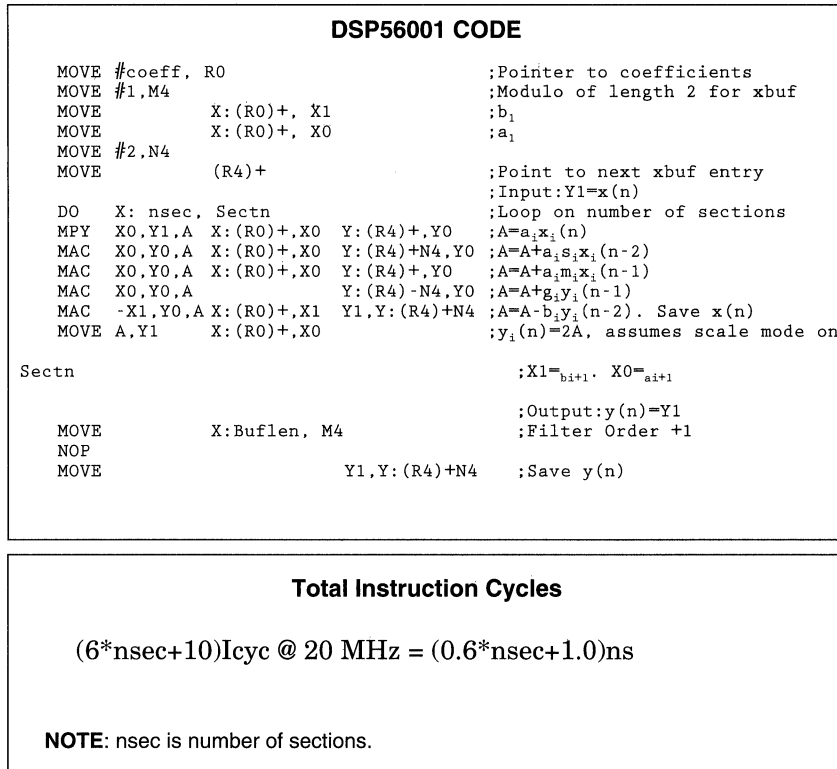


Fig. B.35 DSP56001 Code for Cascaded Direct-Form Filter

B.6 FILTER DESIGN AND ANALYSIS SYSTEM

"The filter example used is a sixth-order Butterworth lowpass filter with a cutoff frequency of approximately 225 Hz and a sample frequency of 1000 Hz."

The following paragraphs discuss the design of a cascaded filter in both the direct-form and canonic implementations, using a software package, QEDesign (formerly FDAS¹), available from Momentum Data Systems, Inc. The filter example used is a sixth-order Butterworth lowpass filter with a cutoff frequency of approximately 225 Hz and a sample frequency of 1000 Hz. Figure B.36 is the log magnitude (gain) plot from the system output. Figure B.37 is the phase as a function of frequency in wrapped format ($-\pi$ wraps to $+\pi$). In addition, Figure B.38 is a zero/pole plot, and Figure B.39 is the group delay, which is the negative of the derivative of the phase with respect to frequency.

1. All references to FDAS in this application note refer to the software package QEDesign.

FDAS will also generate an impulse response, step response, and a linear magnitude plot. The results of the design are written to a file, FDAS.OUT, which contains much useful information. The coefficient data is written to COEFF.FLT. The DSP56001 code generator (MGEN) reads the COEFF.FLT file and generates a DSP56001 assembly source file, COEFF.ASM, which can be assembled by the DSP56001 assembler or linker software. Examples of these files are shown in Figure B.40 to Figure B.45.

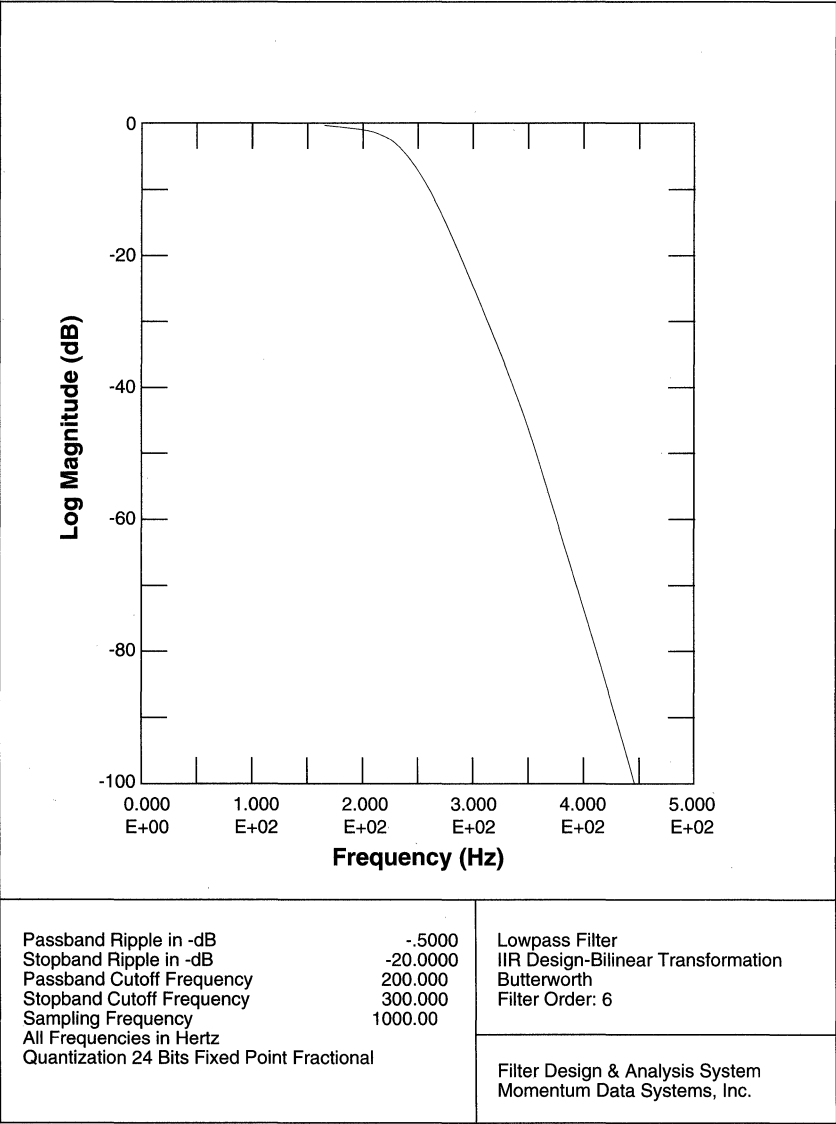


Fig. B.36 Log Magnitude Plot of Example Lowpass Butterworth Filter

B.6.1 Canonic Implementation

Figure B.40 is the output file associated with the FDAS design session, containing information on the analog s-domain equivalent filter as well as the final digital coefficients (listed again in Figure B.41), which have been properly scaled to prevent overflow at the internal nodes and outputs of each cascaded section. This procedure is

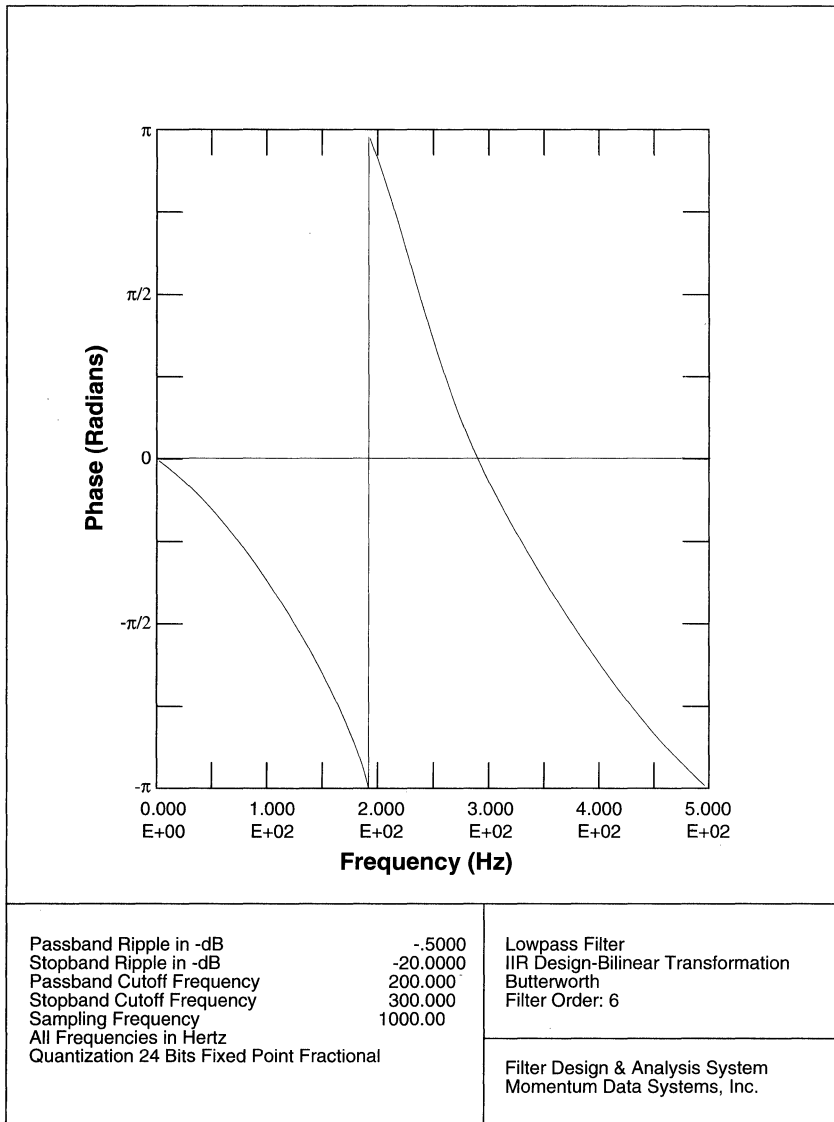


Fig. B.37 Phase Versus Frequency Plot for Example Filter

done automatically by the program in a matter of seconds. Executable code is generated by MGEN (also from Momentum Data Systems, Inc.), as shown in Figure B.42. The code internal to each cascaded section is five instructions long; thus, 500 ns (assuming a DSP56001 clock of 20 MHz) is added to the execution time for each additional second-order section.

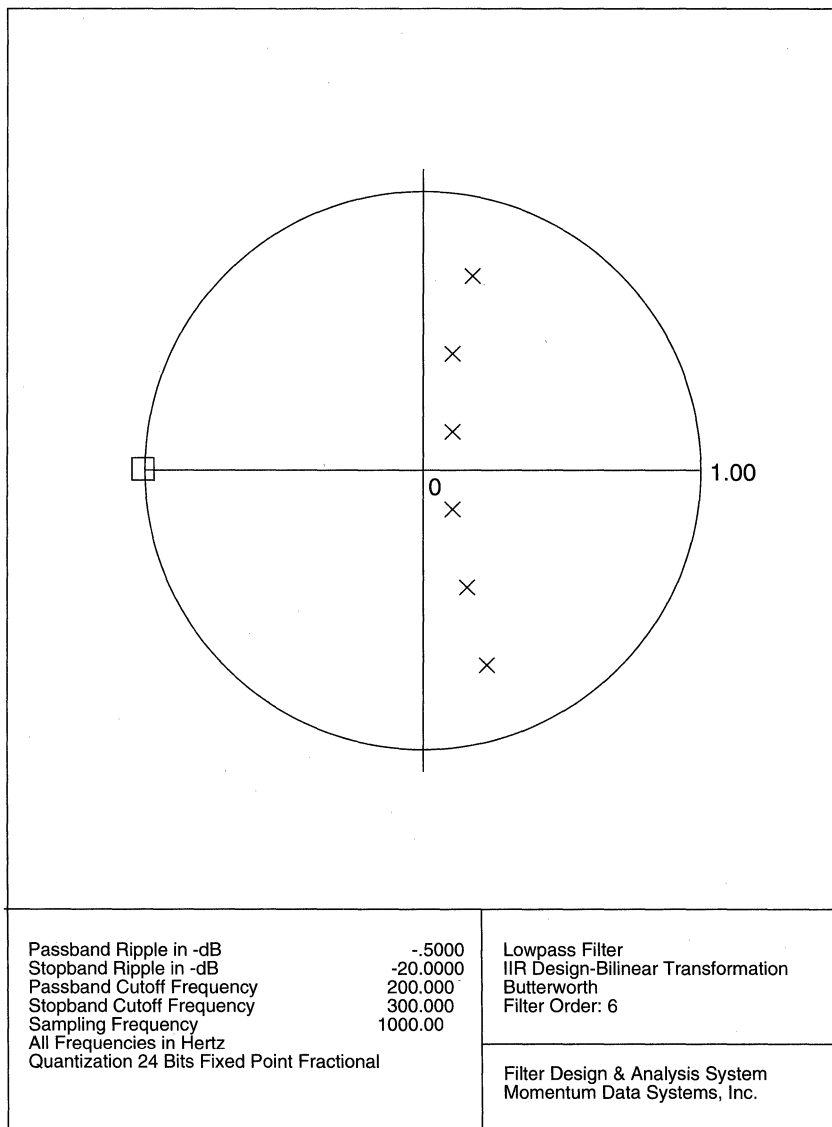


Fig. B.38 Zero/Pole Plot of Sixth-Order Lowpass Example Filter

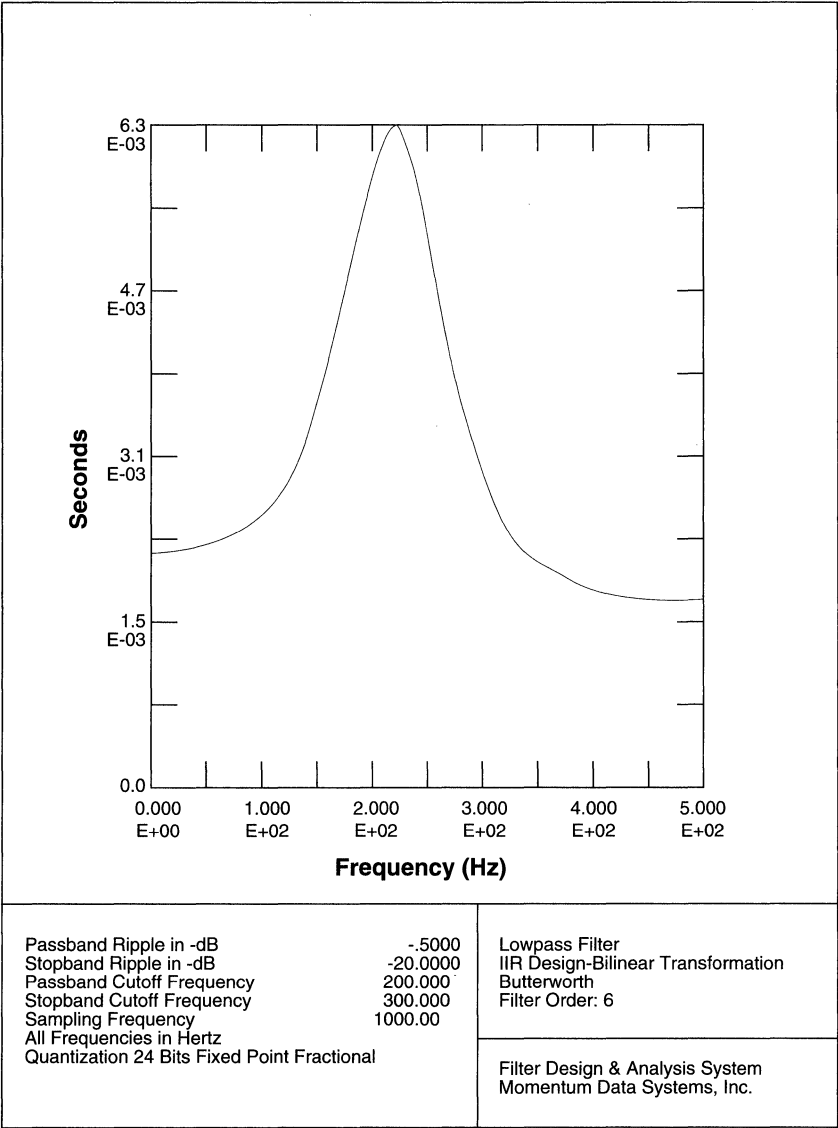


Fig. B.39 Group Delay Versus Frequency for FDAS IIR Example

Fig. B.40 FDAS.OUT File of Example Filter for Cascaded Canonic Implementation

```

Filter Type                      Low Pass
Analog Filter Type              Butterworth
Passband Ripple In -dB          -.5000
Stopband Ripple In -dB          -20.0000
Passband Cutoff Frequency       200.000   Hertz
Stopband Cutoff Frequency       300.000   Hertz
Sampling Frequency              1000.00   Hertz
Filter Order: 6
Filter Design Method: Bilinear Transformation

```

Coefficients of $H_d(z)$ are Quantized to 24 bits

Quantization Type: Fixed Point Fractional

Coefficients Scaled For Cascade Form II

NORMALIZED ANALOG TRANSFER FUNCTION $T(s)$

```

      Numerator Coefficients          Denominator Coefficients
*****
S**2 TERM      S TERM    CONST TERM  S**2 TERM  S TERM    CONST TERM
.000000E+00    .000000E+00 .100000E+01 .100000E+01 .193185E+01 .100000E+01
.000000E+00    .000000E+00 .100000E+01 .100000E+01 .141421E+01 .100000E+01
.000000E+00    .000000E+00 .100000E+01 .100000E+01 .517638E+01 .100000E+01
INITIAL GAIN  1.00000000

```

UNNORMALIZED ANALOG TRANSFER FUNCTION $T(s)$

```

      Numerator Coefficients          Denominator Coefficients
*****
S**2 TERM      S TERM    CONST TERM  S**2 TERM  S TERM    CONST TERM
.000000E+00    .000000E+00 .299809E+07 .100000E+01 .334500E+04 .299809E+07
.000000E+00    .000000E+00 .299809E+07 .100000E+01 .244871E+04 .299809E+07
.000000E+00    .000000E+00 .299809E+07 .100000E+01 .896290E+03 .299809E+07
INITIAL GAIN  1.00000000

```

DIGITAL TRANSFER FUNCTION $H_d(z)$

```

      Numerator Coefficients          Denominator Coefficients
*****
Z**2 TERM      Z TERM    CONST TERM  Z**2 TERM  Z TERM    CONST TERM
.2520353794    .5040707588 .252035379  1.000000  -.1463916302 .0225079060
.2364231348    .4728463888 .236423134  1.000000  -.1684520245 .1765935421
.3606387377    .7212774754 .360638737  1.000000  -.2279487848 .5921629667
INITIAL GAIN  .876116210

```

ZEROES OF TRANSFER FUNCTION $H_d(z)$

Real Part	Imaginary Part	Real Part	Imaginary Part	Radius
-.100000000E+01	.000000000E+00	-.100000000E+01	.000000000E+00	.100000000E+01
-.999290168E+00	.000000000E+00	-.100071034E+01	.000000000E+00	.999290168E+00
-.100000000E+01	.000000000E+00	-.100000000E+01	.000000000E+00	.100000000E+01

POLES OF TRANSFER FUNCTION $H_d(z)$

Real Part	Imaginary Part	Real Part	Imaginary Part	Radius
.731958151E-01	.130959072E+00	.731958151E-01	-.130959072E+00	.150026351E+00
.842260122E-01	.411703195E+00	.842260122E-01	-.411703195E+00	.420230344E+00
.113974392E+00	.761034036E+00	.113974392E+00	-.761034036E+00	.769521258E+00

```

NUMERATOR COEFFICIENTS - HIGHEST ORDER FIRST (in z)

.214893785E-01 .128936282E+00 .322340720E+00 .429787634E+00 .322340720E+00
.128936282E+00 .214893785E-01

DENOMINATOR COEFFICIENTS - HIGHEST ORDER FIRST (in z)

.100000000E+01 -.542792439E+00 .887692610E+00 .267088214E+00 .143235132E+0
-.184597152E-01 .235370025E-02

IMPULSE RESPONSE MIN = -.179722E+00, MAX = .452734E+00

```

Fig. B.41 COEFF.OUT File of Example Filter Design-Scaled for Cascaded Canonic Implementation

```

FILTER COEFFICIENT FILE
IIR DESIGN
FILTER TYPE LOW PASS
ANALOG FILTER TYPE BUTTERWORTH
PASSBAND RIPPLE IN -dB -.5000
STOPBAND RIPPLE IN -dB -20.0000
PASSBAND CUTOFF FREQUENCY .200000E+03 HERTZ
STOPBAND CUTOFF FREQUENCY .300000E+03 HERTZ
SAMPLING FREQUENCY .100000E+04 HERTZ
FILTER DESIGN METHOD: BILINEAR TRANSFORMATION
FILTER ORDER 6 0006h
NUMBER OF SECTIONS 3 0003h
NO. OF QUANTIZED BITS 24 0018h
QUANTIZATION TYPE - FRACTIONAL FIXED POINT
COEFFICIENTS SCALED FOR CASCADE FORM 11
0 00000000 /* shift count for overall gain */
7349395 00702493 /* overall gain */
0 00000000 /* shift count for section 1 value */
2114226 002042B2 /* section1 coefficient B0 */
4228452 00408564 /* section1 coefficient B1 */
2114226 002042B2 /* section1 coefficient B2 */
1228022 0012BCF6 /* section1 coefficient A1 */
-188810 FFD1E76 /* section1 coefficient A2 */
0 00000000 /* shift count for section 2 values */
1983261 001E431D /* section2 coefficient B0 */
3966523 003C863B /* section2 coefficient B1 */
1983261 001E431D /* section2 coefficient B2 */
1413078 00158FD6 /* section2 coefficient A1 */
-1481374 FFE96562 /* section2 coefficient A2 */
0 00000000 /* shift count for section 3 values */
3025257 002E2969 /* section3 coefficient B0 */
6050514 005C52D2 /* section3 coefficient B1 */
3025257 002E2969 /* section3 coefficient B2 */
1912173 001D2D6D /* section3 coefficient A1 */
-4967423 FFB43401 /* section3 coefficient A2 */
.2520353794097900D+00 3FD0215900000000 .25203538E+00 /* section 1 B0 */
.504D070588195801D+00 3FE0215900000000 .50407076E+00 /* section 1 B1 */
.2520353794097900D+00 3FDD215900000000 .25203538E+00 /* section 1 B2 */
.1463916301727295D+00 3FC2BCF600000000 .14639171E+00 /* section 1 A1 */
-.2250790596008301D-01 BF970C5000000000 -.22507921E-01 /* section 1 A2 */
.2364231348037720D+00 3FCE431D00000000 .23642325E+00 /* section 2 B0 */
.4728463888168335D+00 3FDE431D80000000 .47284651E+00 /* section 2 B1 */
.2364231348037720D+00 3FCE431D00000000 .23642325E+00 /* section 2 B2 */
.1684520244598389D+00 03FC58FD60000000 .168452D4E+D0 /* section 2 A1 */
.17659354Z0989990D+00 BFC69A9E000000D0 -.17659356E+00 /* section 2 A2 */
.3606387376785278D+00 3FD714B480000000 .36063876E+00 /* section 3 B0 */

```



```

.7212774753570557D+00 3FE714B480000000 .72127753E+00 /* section 3 B1 */
.3606387376785278D+00 3FD714B480000000 .36063876E+00 /* section 3 B2 */
.2279487848281860D+00 3FCD2D6DD0000000 .22794882E*00 /* section 3 A1 */
-.5921629667282104D+00 BFE2F2FFC0000000 -.59216306E+00 /* section 3 A2 */

```

Fig. B.42 COEFFASM File Generated by MGEN for Example Design and Cascaded Canonic Implementation

```

1      COEFF          ident 1,0
2                      include 'head2.asm'
3
4                      page 132, 66,          0.10
5                      pt      cex,          mex
6
7      ; This program implements an IIR filter in cascaded canonic sections
8      ; The coefficients of each section are scaled for cascaded canonic sections
9
10
11 00FFFF datin      equ $ffff                ;location in Y memory of input file
12 00FFFF datout     equ $ffff                ;location in Y memory of output file
13 00FFFF0 m_scr      equ $ffff0              ;sci control register
14 00FFFF1 m_ssr      equ $ffff1              ;sci status register
15 00FFFF2 m_sccr     equ $ffff2              ;sci clock control register
16 00FFFF1 m_pcc      equ $fff1               ;port c control register
17 00FFFF m_ipr       equ $ffff              ;interrupt priority register
18 000140 xx          equ @cvi(20480/64*1.000)) ;timer interrupt value
19 FFD8F1 m_tim       equ -9999              ;board timer interrupt value
20 000003 nsec        equ 3                  ;number of second order sections
21                      include 'cascade2.asm'
22
23 ; This code segment implements cascaded biquad sections in canonic form
24 ;
25
26 cascade2 macro nsec
27 m
28 m
29 m
30 m
31 m
32 m
33 m
34 m
35 m
36 m
37 m
38 m
39 m
40
41 ;
42 ;multiple shift left macro
43
44 mshl      macro      scount
45 m          if
46 m          rep      #scount
47 m          asl      a
48 m          endif
49 m          endm
50 X:0000          org      x:0
51 X:0000          states   ds      2*nsec
52 Y:0000          org      y:0
53                  coef
54 d      Y:0000 FE8F38      dc      $FE8F3B      ;a(*,2)/2      =-.01125395 section number 1
55 d      Y:0001 095E7B      dc      $095E7B      ;a(*,1)/2      =.07319582 section number 1
56 d      Y:0002 102159      dc      $102159      ;b(*,2)/2      =.12601769 section number 1
57 d      Y:0003 2042B2      dc      $2042B2      ;b(*,1)/2      =.25203538 section number 1
58 d      Y:0004 D02159      dc      $D02159      ;b(*,0)/2-0.5 =-.37398231 section number 1
59 d      Y:0005 F4B2B1      dc      $F4B2B1      ;a(*,2)/2      =-.08829677 section number 2
60 d      Y:0006 0AC7EB      dc      $0AC7EB      ;a(*,1)/2      =.08422601 section number 2
61 d      Y:0007 0F218E      dc      $0F218E      ;b(*,2)/2      =.11821157 section number 2
62 d      Y:0008 1E4313      dc      $1E431D      ;b(*,1)/2      =.23642319 section number 2
63 d      Y:0009 CF218E      dc      $CF218E      ;b(*,0)/2-0.5 =-.38178843 section number 2
64 d      Y:000A DA1A01      dc      $DA1A01      ;a(*,2)/2      =-.29608148 section number 3
65 d      Y:000B 0E96B6      dc      $0E96B6      ;a(*,1)/2      =.11397439 section number 3
66 d      Y:000C 1714B4      dc      $1714B4      ;b(*,2)/2      =.18031937 section number 3
67 d      Y:000D 2E2969      dc      $2E2969      ;b(*,1)/2      =.36063874 section number 3
68 d      Y:000E D714B4      dc      $0714B4      ;b(*,0)/2-0.5 =-.31968063 section number 3
69 Y:0008          org      y:200

```

```

70 d Y:00C8 381249 igain dc 3674697
71 000000 scount equ 0 ;final shift count
72 include 'bodyZ.asm'
73
74 000040 start equ $40 ;origin for user program
75
76 P:0000 org p:$0 ;origin for reset vector
77 P:0000 0C0040 jmp start ;jump to 'start' on system reset
78
79 P:001c org p:$1c ;origin for timer interrupt vector
80 P:001c 0BF080 jsr filter ;jump to 'filter' on timer interrupt
81 000051
82
83 P:0040 org p:start ;origin for user program
84 P:0040 0003F8 ori #3,mr ;disable all interrupts
85 P:0041 08F4B2 movep #xx-1,x:m_sccr ;cd=xx-1 for divide by xx
86 00013F ;
87 P:0043 08F4A1 movep #$7,x:m_pcc ;set cc(2;0) to turn on timer
88 000007 ;
89 P:0045 08F4B0 movep #$2000,x:m_scr ;enable timer interrupts
90 002000 ;
91 P:0047 08F4BF movep #$c000,x:m_ipr ;set interrupt priority for sci
92 00C000 ;
93
94 P:0049 300000 move #states,r0 ;initialize internal state storage
95 P:004A 200013 clr a ;* set memory to zero
96 P:004B 0606A0 rep #nsec*2 ;*
97 P:004C 565800 move a,x:(r0)+ ;*
98
99 P:004D 4FF000 move y:igain,y1 ;yl=initial gain/2
100 0000C8 ;
101 P:004F 00FCB8 andi #$fc,mr ;allow interrupts
102 P:0050 0C0050 jmp * ;wait for interrupt
103
104 filter
105 ori #$08,mr ;set scaling node
106 move #states,r0 ;point to filter states
107 move #coef,r4 ;point to filter coefficients
108 movep y:datin,y0 ;get sample
109
110 cascade2 nsec ;do cascaded biquads
111 ;
112 ; assumes each section's coefficients are divided by 2
113 ;
114
115 P:0055 F098B0 mpy y0,y1,a x:(r0)+x0 y:(r4)+y0 ;x0=1stsection w(n-2),y0=ai2/2
116 P:0056 060380 do #nsec,_ends ;do each section
117 00005C ;
118
119 P:0058 F490D2 mac x0,y0,a x:(r0)-,x1 y:(r4)+y0 ;x1=w(n-1)y0=ail/2
120 P:0059 F418E3 macr x1,y0,a x1,x:(r0)+ y:(r4)+y0 ;push w(n-1) to w(n 2),y0=bi2/2
121 P:005A F800D2 mac x0,y0,a a,x:(r0) y:(r4)+y0 ;push w(n) to w(n-1),y0=bi1/2
122 P:005B F098E2 mac x1,y0,a x:(r0)+,x0 y:(r4)+y0 ;get this iter w(n),y0=bi0/2
123 P:005C F098D2 mac x0,y0,a x:(r0)+,x0 y:(r4)+y0 ;next iter:x0=w(n-2),y0=ai2/2
124 _ends
125 mshl scount
126 if scount
127 endif
128 rnd a ;round result
129 movep a,y:datout ;output sample
130 rti
131
132 end

```

0 Errors
0 Warnings

B.6.2 Transpose Implementation (Direct Form I)

Figure B.43 is the output file from FDAS for a transpose-form implementation. As before, it contains information on the analog s-domain equivalent filter as well as the final digital coefficients (listed again in Figure B.44), which have been properly scaled to prevent overflow at the internal nodes and outputs of each cascaded section. Because of the stability of the internal node gain of the transpose form and because of the response of each second-order Butterworth section, scaling was not done by the

program because it was not needed. Executable code shown in Figure B.45 is again generated by MGEN. The code internal to each cascaded section is seven instructions long; thus, 700 ns (assuming a DSP56001 clock of 20 MHz) is added to the execution time for each additional second-order section.

Fig. B.43 FDAS.OUT File of Example Filter for Cascaded Transpose-Form Implementation

```

FILTER TYPE          LOW PASS
ANALOG FILTER TYPE   BUTTERWORTH
PASSBAND RIPPLE IN -dB      -5.0000
STOPBAND RIPPLE IN -dB      -20.0000
PASSBAND CUTOFF FREQUENCY   200.000 HERTZ
STOPBAND CUTOFF FREQUENCY   300.000 HERTZ
SAMPLING FREQUENCY         1000.00 HERTZ
FILTER ORDER: 6
FILTER DESIGN METHOD: BILINEAR TRANSFORMATION
COEFFICIENTS OF Hd(Z) ARE QUANTIZED TO 24 BITS
QUANTIZATION TYPE: FIXED POINT FRACTIONAL
COEFFICIENTS SCALED FOR CASCADE FORM I (TRANPOSE FORM)

      NORMALIZED ANALOG TRANSFER FUNCTION T(s)

      NUMERATOR COEFFICIENTS
      *****
      S**2 TERM  S TERM  CONST TERM
      .000000E+00 .000000E+00 .100000E+01
      .000000E+00 .000000E+00 .100000E+01
      .000000E+00 .000000E+00 .100000E+01
      INITIAL GAIN 1.00000000

      DENOMINATOR COEFFICIENTS
      *****
      S**2 TERM  S TERM  CONST TERM
      .100000E+01 .193185E+01 .100000E+01
      .100000E+01 .141421E+01 .100000E+01
      .100000E+01 .517638E+00 .100000E+01

      UNNORMALIZED ANALOG TRANSFER FUNCTION T(s)

      NUMERATOR COEFFICIENTS
      *****
      S**2 TERM  S TERM  CONST TERM
      .000000E+00 .000000E+00 .299809E+07
      .000000E+00 .000000E+00 .299809E+07
      .000000E+00 .000000E+00 .299809E+07
      INITIAL GAIN 1.00000000

      DENOMINATOR COEFFICIENTS
      *****
      S**2 TERM  S TERM  CONST TERM
      .100000E+01 .334500E+04 .299809E+07
      .100000E+01 .244871E+04 .299809E+07
      .100000E+01 .896290E+03 .299809E+07

      DIGITAL TRANSFER FUNCTION Hd(z)

      NUMERATOR COEFFICIENTS
      *****
      Z**2 TERM  Z TERM  CONST TERM
      .2190289497 .4380580187 .2190289497
      .2520353794 .5040707588 .2520353794
      .3410534859 .6821070910 .3410534859
      INITIAL GAIN 1.00000000

      DENOMINATOR COEFFICIENTS
      *****
      Z**2 TERM  Z TERM  CONST TERM
      1.000000 - .1463916302 .0225079060
      1.000000 - .1684520245 .1765935421
      1.000000 - .2279487848 .5921629667

      ZEROES OF TRANSFER FUNCTION Hd(z)

      REAL PART  IMAGINARY PART  REAL PART  IMAGINARY PART  RADIUS
      -.999262530E+00 .000000000E+00 -.100073501E+01 .000000000E+00 .999262530E+00
      -.100000000E+01 .000000000E+00 -.100000000E+01 .000000000E+00 .100000000E+01
      -.999408962E+00 .000000000E+00 -.100059139E+01 .000000000E+00 .999408962E+00

      POLES OF TRANSFER FUNCTION Hd(z)

      REAL PART  IMAGINARY PART  REAL PART  IMAGINARY PART  RADIUS
      -.731958151E-01 -.130959072E+00 -.731958151E-01 -.130959072E+00 -.150026351E+00
      -.842260122E-01 -.411703195E+00 -.842260122E-01 -.411703195E+00 -.420230344E+00
      -.113974392E+00 -.761034036E+00 -.113974392E-00 -.761034036E+00 -.769521258E+00

```

```

NUMERATOR COEFFICIENTS - HIGHEST ORDER FIRST (in z)

.188271907E-01 .112963161E+00 .282407928E+00 .376543916E+00 .282407928E+00
.112963161E+00 .188271907E-01

DENOMINATOR COEFFICIENTS - HIGHEST ORDER FIRST (in z)

.10000000E+01 -.542792439E+00 .887692610E+00 -.267088214E+00 .143235132E+00
.184597152E-01 .235370025E-02

IMPULSE RESPONSE MIN = -.179722E+00, MAX = .452734E+00

```

Fig. B.44 COEFF.FLT File for Example Filter Design—Scaled for Cascaded Transpose Form

```

FILTER COEFFICIENT FILE
IIR DESIGN
FILTER TYPE LOW PASS
ANALOG FILTER TYPE BUTTERWORTH
PASSBAND RIPPLE IN -dB -.5000
STOPBAND RIPPLE IN -dB -20.0000
PASSBAND CUTOFF FREQUENCY .200000E+03 HERTZ
STOPBAND CUTOFF FREQUENCY .300000E+03 HERTZ
SAMPLING FREQUENCY .100000E+04 HERTZ
FILTER DESIGN METHOD: BILINEAR TRANSFORMATION
FILTER ORDER 6 0006h
NUMBER OF SECTIONS 3 0003h
NO. OF QUANTIZED BITS 24 0018h
QUANTIZATION TYPE - FRACTIONAL FIXED POINT
COEFFICIENTS SCALED FOR CASCADE FORM I

1 00000001 /* shift count for overall gain */
4194304 00400000 /* overall gain */
0 00000000 /* shift count for section 1 values */
1837348 001C0924 /* section 1 coefficient B0 */
3674697 00381249 /* section 1 coefficient B1 */
1837348 001C0924 /* section 1 coefficient B2 */
1228022 00128CF6 /* section 1 coefficient A1 */
-188810 FFFD1E76 /* section 1 coefficient A2 */
0 00000000 /* shift count for section 2 values */
2114226 00204292 /* section 2 coefficient B0 */
4228452 00408564 /* section 2 coefficient B1 */
2114226 00204282 /* section 2 coefficient B2 */
1413078 00158FD6 /* section 2 coefficient A1 */
-1481374 FFE96562 /* section 2 coefficient A2 */
0 00000000 /* shift count for section 3 values */
2860964 002BA7A4 /* section 3 coefficient B0 */
5721929 00574F49 /* section 3 coefficient B1 */
2860964 002BA7A4 /* section 3 coefficient B2 */
1912173 001D2D6D /* section 3 coefficient A1 */
-4967423 FFB43401 /* section 3 coefficient A2 */
.2190289497375488D+00 3FCC092400000000 .21902905E+00 /* section 1 B0 */
.4380580186843872D+00 3FDC092480000000 .43805810E+00 /* section 1 B1 */
.2190289497375488D+00 3FCC092400000000 .21902905E+00 /* section 1 B2 */
.146391630172 n95D+00 3FC2BCF600000000 .14639171E+00 /* section 1 A1 */
-.2250790596008301D-01 BF970C5000000000 -.22507921E-01 /* section 1 A2 */
.2520353794097900D+00 3FD0215900000000 .25203538E+00 /* section 2 B0 */
.5040707588195801D+00 3FE0215900000000 .50407076E+00 /* section 2 B1 */
.2520353794097900D+00 3FD0215900000000 .252D3538E+00 /* section 2 B2 */
.1684520244598389D+00 3FC58FD600000000 .16845204E+00 /* section 2 A1 */
-.1765935420989990D+00 BFC69A9E00000000 -.17659356E+00 /* section 2 A2 */
.34105348587036130+00 3FDD3D2000000000 .34105356E+00 /* section 3 B0 */
.6821070909500122D+00 3FED3DZ400000000 .68210712E+00 /* section 3 B1 */

```

```
.3410534858703613D+00 3FDS3D200000000 .34105356E+00 /* section 3 B2 */
.2279487848281860D+00 3FCD2D6D00000000 .22794882E+00 /* section 3 A1 */
-.5921629667282104D+00 BFE2F2FFC0000000 -.59216306E+00 /* section 3 A2 */
```

Fig. B.45 COEFF.ASM File Generated by MGEN for Example Design and Cascaded Transpose-Form Implementation

```
1      COEFF      ident 1.0
2                  include 'head1.asm'
3
4                  page132,66,0,10
5                  optcex, mex
6
7      ;
8      ; This program implements an IIR filter in cascaded transpose sections
9      ; The coefficients of each section are scaled for cascaded transpose sections
10
11      00FFFF      datin      equ      $ffff      ;location in Y memory of input file
12      00FFFF      datout     equ      $ffff      ;location in Y memory of output file
13      00FFF0      m_scr      equ      $fff0      ;sci control register
14      00FFF1      m_ssr      equ      $fff1      ;sci status register
15      00FFF2      m_sccr     equ      $fff2      ;sci clock control register
16      00FFE1      m_pcc      equ      $ffe1      ;port c control register
17      00FFFF      m_ipr      equ      $fff      ;interrupt priority register
18      000140      xx         equ      @cvi(20480/(64*1.000)) ;timer interrupt value
19      FFD8F1      m_tim      equ      -9999      ;board timer interrupt value
20      000003      nsec       equ      3          ;number of second order sections
21                  include 'cascadel.asm'
22
23      ;
24      ;This code segment implements cascaded biquad sections in transpose form
25
26      cascadel      macro      nsec
27
28      m      ;assumes each section's coefficients are divided by 2
29      m
30      m      do      #nsec,_ends      ;do each section
31      m      macr      y0.y1,a      x:(r1),b      y:(r4)+,y0      ;a=x(n)*bi0/2+wi1/2,b=wi2,y0=bi1/2
32      m      asr      b      a,x0      ;b=wi2/2,x0=y(n)
33      m      mac      y0.y1,b      y:(r4)+,y0      ;b=x(n)*bi1/2+wi2/2,y0=ai1/2
34      m      macr      x0.y0,b      y:(r4)+,y0      ;b=bty(n)*ai1/2,y0=bi2/2
35      m      mpy      y0.y1,b      b,x:(r0)+      y:(r4)+,y0      ;b=x(n)*bi2/2,save wi1,y0=ai2
36      m      macr      x0.y0,b      x:(r0),a      a,y1      ;b=bty(n)*ai2/2,a=next iter wi1,
37      m      ;yl=output of section i
38      m      asr      a      b,x:(r1)+      y:(r4)+,y0      ;a=next iter wi1/2,save wi2,
39      m      ;y0=next iter bi0
40      m      _ends
41      m      endm
42
43
44      X:0000      org      x:0
45      X:0000      statel     dsm      nsec
46      X:0004      state2     dsm      nsec
47      Y:0000      org      y:0
48
49      coef
50      d Y:0000      0E0492      dc      $0E0492      ;b(*,0)/2      = .10951447 section number 1
51      d Y:0001      1C0924      dc      $1C0924      ;b(*,1)/2      = .21902901 section number 1
52      d Y:0002      095E7B      dc      $095E7B      ;a(*,1)/2      = .07319582 section number 1
53      d Y:0003      0E0492      dc      $0E0492      ;b(*,2)/2      = .10951447 section number 1
54      d Y:0004      FE8F3B      dc      $FE8F3B      ;a(*,2)/2      = -.01125395 section number 1
55      d Y:0005      102159      dc      $102159      ;b(*,0)/2      = .12601769 section number 2
56      d Y:0006      2042B2      dc      $2042B2      ;b(*,1)/2      = .25203538 section number 2
57      d Y:0007      0AC7EB      dc      $0AC7EB      ;a(*,1)/2      = .08422601 section number 2
58      d Y:0008      102159      dc      $102159      ;b(*,2)/2      = .12601769 section number 2
59      d Y:0009      F4B2B1      dc      $F4B2B1      ;a(*,2)/2      = -.08829677 section number 2
60      d Y:000A      1SD3D2      dc      $1SD3D2      ;b(*,0)/2      = .17052674 section number 3
61      d Y:000B      2BA7A4      dc      $2BA7A4      ;b(*,1)/2      = .34105355 section number 3
62      d Y:000C      0E9686      dc      $0E9686      ;a(*,1)/2      = .11397439 section number 3
63      d Y:000D      1SD3D2      dc      $1SD3D2      ;b(*,2)/2      = .17052674 section number 3
64      d Y:000E      DA1A01      dc      $DA1A01      ;a(*,2)/2      = -.29608148 section number 3
65                  include 'body1.asm'
66
67      000040      start      equ      $40          ;origin for user program
68
69      P:0000      org      p:$0          ;origin for reset vector
70      P:0000      0C0040      jmp      start      ;jump to 'start' on system reset
71
72      P:001C      org      p:$1c          ;origin for timer interrupt vector
```

```

72   P:001C   0BF080   jsr   filter           ;jump to 'filter' on timer interrupt
      000057
73
74   P:0040           org   p:start           ;origin for user program
75
76   P:0040   0003F8   ori    #3,mr           ;disable all interrupts
77   P:0041   08F4B2   movep   #(xx-1),x:m_sccr   ;cd=xx 1 for divide by xx
      00013F
78   P:0043   08F4A1   movep   #$7,x:m_pcc      ;set cc(2;0) to turn on timer
      000007
79   P:0045   08F4B0   movep   #$2000,x:m_scr   ;enable timer interrupts
      002000
80   P:0047   08F4BF   movep   #$c000,x:m_ipr   ;set interrupt priority for sci
      00C000
81
82
83   P:0049   300000   move    #statel,r0       ;point to filter statel
84   P:004A   310400   move    #state2,r1      ;point to filter state2
85   P:004B   340000   move    #coef,r4        ;point to filter coefficients
86   P:004C   0502A0   move    #nsec 1,m0      ;addressing modulo nsec
87   P:004D   050EA4   move    #5*nsec-1,m4    ;addressing modulo 5*nsec
88   P:004E   0502A1   move    #nsec-1,m1      ;addressing modulo nsec
89   P:004F   200013   clr     a               ;initialize internal state storage
90   P:0050   0603A0   rep     #nsec           ;* zero statel
91   P:0051   565800   move    a,x:(r0)+       ;*
92   P:0052   0603A0   rep     #nsec           ;* zero state2
93   P:0053   565900   move    a,x:(r1)+       ;*
94
95   P:0054   F88000   move    x:(r0),a   y:(r4)+,y0   ;a=w1 (initially zero) ,y0=b10/2
96
97   P:0055   00FCB8   andi    #fsc,mr        ;allow interrupts
98   P:0056   0C0056   jmp     *              ;wait for interrupt
99
100                                     filter
101   P:0057   0008F8   ori     #08,mr         ;set scaling mode
102   P:0058   09473F   movep   y:datin,y1      ;get sample
103
104                                     cascadel nsec           ;do cascaded biquads
105 +
106 +                                     ; assumes each section's coefficients are divided by 2
107 +
108 + P:0059   060380   do      #nsec,_ends    ;do each section
      000061
109 + P:005B   FC81B3   macr    y0,y1,a   x:(r1),b   y:(r4)+,y0   ;a=x(n)*bi0/2+wi1/2,b=wi2.y0=bi1/2
110 + P:005C   21C42A   asr     b           ;b=wi2/2,x0=y(n)
111 + P:005D   4EDCBA   mac     y0,y1,b       y:(r4)+,y0   ;b=x(n)*bi1/2+wi2/2,y0=ai1/2
112 + P:005E   4EDCDB   macr    x0,y0,b       y:(r4)+,y0   ;b=bty(n)*ai1/2,y0=bi2/2
113 + P:005F   FC18B8   mpy     y0,y1,b   b,x:(r0)+   y:(r4)+,y0   ;b=x(n)*bi2/2,save wi1,y0=ai2
114 + P:0060   19A0DB   macr    x0,y0,b   x:(r0),a   a,y1   ;b=bty(n)*ai2/2,a=next iter wi1,
115 +                                     ;y1=output of section i
116 + P:0061   FC1922   asr     a           ;a=next iter wi1/2,save wi2,
117 +                                     ;y0=next iter bi0
118 + _ends
119
120   P:0D62   09C73F   movep   y1,y:datout     ;output sample
121   _endp
122
123   P:0063   000004   rti
124   end
0   Errors
0   Warnings

```

B.7 FIR FILTERS

"The filter can be efficiently implemented on the DSP56001 by using modulo addressing to implement the shifting and parallel data moves to load the multiplier-accumulator."

If phase distortion is of secondary importance to magnitude response, the desired filter can generally be implemented with less memory, less computational complexity, and at the lowest cost using IIR structures. On the other hand, in applications requiring linear phase in the passband and a specific magnitude response, the

specified filter is generally best implemented using FIR structures. Examples of applications requiring linear phase are as follows:

1. Communication systems such as modems or Integrated Services Digital Networks (ISDN) in which the data pulse shape and relative timing in the channel must be preserved
2. Ideal differentiators which provide a 90-degree phase shift at all frequencies in addition to a constant group delay
3. Hilbert transformers used to demodulate complex signals such as those used in high-speed modems
4. Hi-fidelity audio systems in which phase distortion of recorded music must be minimized to reproduce the original sound with as much fidelity as possible
5. System synthesis in which the system impulse response is known as *a priori*. FIR filters are also important because they are all-zero filters (i.e., no feedback) and are therefore guaranteed to be stable.

B.7.1 Linear-Phase FIR Filter Structure

The basic structure of a FIR filter is simply a tapped delay line in which the output from each tap is summed to generate the filter output. This is shown in Figure B.46. This structure can be represented mathematically as:

$$y(n) = \sum_{i=0}^{N-1} h(i)x(n-i) \quad (\text{B.35})$$

where: $x(n)$ is the most recent ($t = nT$) input signal sample

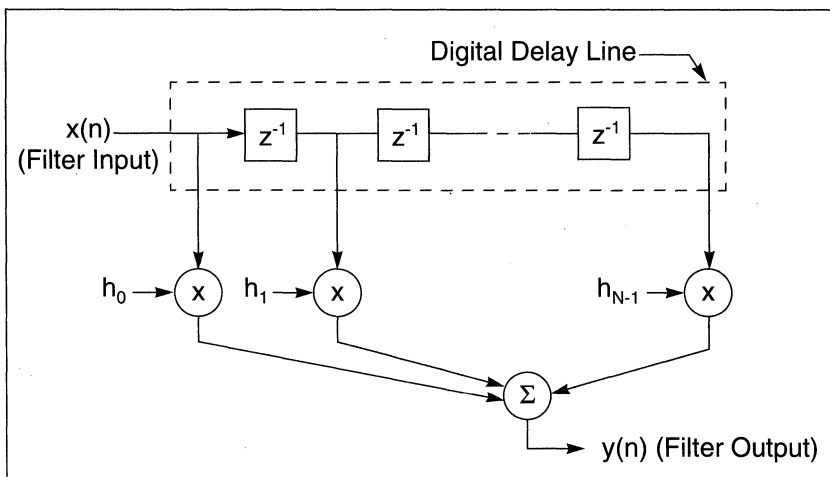


Fig. B.46 FIR Structure

$x(n-i)$ is signal samples delayed by i sample periods (iT)

$h(i)$ is the tap weights (or filter coefficients)

$y(n)$ is the filter output at time $t = nT$

From this structure, it is easy to see why the filter is termed finite—the impulse response of the filter will be identically zero after N sample periods because an impulse input (i.e., $x(n) = 1$, $x(n-i) = 0$ for $i \neq 0$) will have traversed the entire delay line at time $t = NT$. That is, the impulse response of FIR filters will only last for a finite period of time. This response is in contrast to IIR filters which will “ring” in response to an impulse for an infinite period of time. The values of the coefficients represent the impulse response of the FIR filter as can be seen by evaluating Eqn. (B.35) over N sample periods for a single-unit input pulse at time $t = 0$.

There are no feedback terms in the structure (i.e., Eqn. (B.35) has no denominator) but rather only N zeros. By taking the z -transform of Eqn. (B.35), the transfer function of the filter becomes:

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{i=0}^{N-1} h(i)z^{-i} \quad (\text{B.36})$$

Eqn. (B.36) is a polynomial in z of order N . The roots of this polynomial are the N zeros of the filter.

The same procedures used to calculate the magnitude and phase response of IIR filters can be applied to FIR filters. Accordingly, the gain, $G(\theta)$, can be obtained by substituting $z = e^{j\theta}$ into Eqn. (B.36) (where $\theta = 2\pi/f_s$ is the normalized digital frequency), then taking the absolute value so that:

$$\begin{aligned} G(\theta) &= \left| \sum_{i=0}^{N-1} h(i)e^{-ji\theta} \right| \\ &= \sum_{i=0}^{N-1} [h(i)\cos i\theta - jh(i)\sin i\theta] \end{aligned} \quad (\text{B.37})$$

The phase response, $\phi(\theta)$, is found by taking the inverse tangent of the ratio of the imaginary to real components of $G(\theta)$ so that:

$$\phi(\theta) = \tan^{-1} \frac{\sum_{i=0}^{N-1} h(i)\sin i\theta}{\sum_{i=0}^{N-1} h(i)\cos i\theta} \quad (\text{B.38})$$

where: $h(i)$ is implicitly assumed to be real

Intuitively, it can be reasoned that, for any pulse to retain its general shape and relative timing (i.e., pulse width and time delay to its peak value) after passing through a filter, the delay of each frequency component making up the pulse must be the same so that each component recombines in phase to reconstruct the original

shape. In terms of phase, this implies that the phase delay must be linearly related to frequency or:

$$\phi(\theta) = -\tau\theta \quad (\text{B.39})$$

This relationship is illustrated in Figure B.47 in which a pulse having two components, $\sin \omega t$ and $\sin 2\omega t$, passes through a linear-phase filter having a delay of two cycles per Hz and a constant magnitude response equal to unity. That is, the fundamental is delayed by two cycles (4π) and the first harmonic is delayed by four cycles (8π). The group delay of a system, τ_g , is defined by taking the derivative of $\phi(\theta)$:

$$\tau_g \equiv -\frac{d\phi}{d\theta} \quad (\text{B.40})$$

Since θ is the normalized frequency, τ_g in Eqn. (B.40) is a dimensionless quantity and can be related to the group delay in seconds by dividing by the sample frequency, f_s . For a linear-phase system, τ_g is independent of θ and is equal to τ . This fact can be seen by substituting Eqn. (B.39) into Eqn. (B.40).

In this example, τ_g is two cycles per Hz. Note that the pulse shape within the filter has been retained but is twice the width of the original pulse; that is the change in the pulse width is equal to the group delay. The negative sign indicates the phase is retarded or delayed (i.e., a causal system).

The impact of the requirement of linear phase on the design of a FIR filter can be seen by substituting Eqn. (B.38) into Eqn. (B.39) so that:

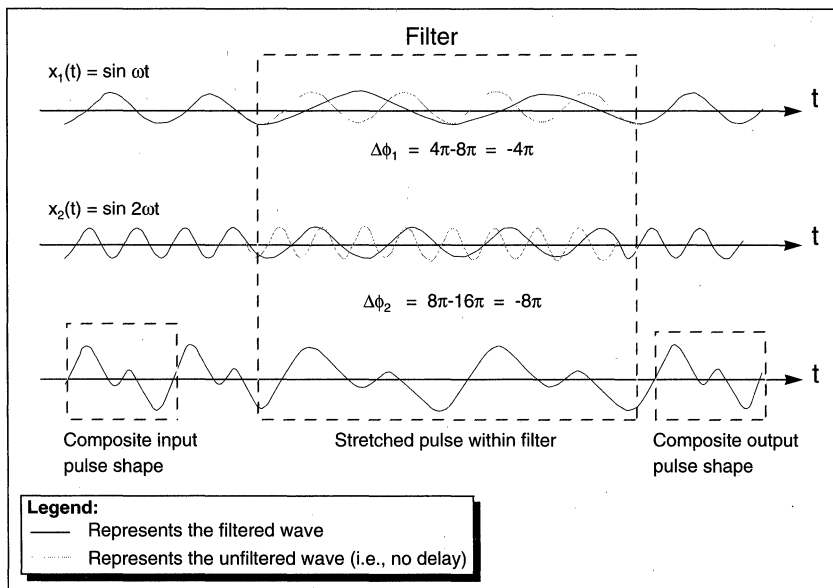


Fig. B.47 Signal Data through a FIR Filter

$$\tan \theta \tau_g = \frac{\sum_{i=0}^{N-1} h(i) \sin i \theta}{\sum_{i=0}^{N-1} h(i) \cos i \theta} \quad (\text{B.41})$$

which can be written as:

$$\sum_{i=0}^{N-1} h(i) (\cos i \theta \sin \theta \tau_g - \sin i \theta \cos \theta \tau_g) = 0$$

or

$$\sum_{i=0}^{N-1} h(i) \sin[\theta(\tau_g - i)] = 0 \quad (\text{B.42})$$

The solution to Eqn. (B.42) is the constraint on τ_g for a FIR filter to be linear phase. The solution to Eqn. (B.42) can be found by expanding the left-hand side (LHS) as follows:

$$\begin{aligned} \text{LHS} = & h_0 \sin \theta \tau_g + h_1 \sin \theta (\tau_g - 1) + h_2 \sin \theta (\tau_g - 2) + \dots \\ & + h_{N-3} \sin \theta (\tau_g - N + 3) + h_{N-2} \sin \theta (\tau_g - N + 2) + h_{N-1} \sin \theta (\tau_g - N + 1) \end{aligned}$$

so that if:

$$\tau_g = \frac{(N-1)}{2} \quad (\text{B.43})$$

then for every positive argument there will be a corresponding negative argument. For example, the argument for the h_1 term becomes:

$$\theta \left(\frac{N-1}{2} - 1 \right) = \theta \left(\frac{N-3}{2} \right)$$

which is the negative of the argument for the h_{N-2} term, i.e.,

$$\theta \left(\frac{N-1}{2} - N + 2 \right) = -\theta \left(\frac{N-3}{2} \right)$$

so that if:

$$h(i) = h(N-1-i) \quad \text{for } 0 \leq i \leq N-1 \quad (\text{B.44})$$

then Eqn. (B.43) and Eqn. (B.44) represent the solution to Eqn. (B.42). By substituting Eqn. (B.43) into Eqn. (B.39), the phase of a linear-phase FIR filter is given by:

$$\phi(\theta) = -\left(\frac{N-1}{2} \right) \theta \quad (\text{B.45})$$

Therefore, a nonrecursive filter (FIR), unlike a recursive filter (IIR), can have a constant time delay for all frequencies over the entire range (from 0 to $f_s/2$). It is only necessary that the coefficients (and therefore impulse response) be symmetrical about the midpoint between samples $(N-2)/2$ and $N/2$ for even N or about sample $(N-1)/2$ for odd N [see Eqn. (B.44)]. When this symmetry exists, τ_g for the filter will be

$$\left(\frac{N-1}{2}\right) T \text{ seconds.}$$

B.7.2 Linear-Phase FIR Filter Design Using the Frequency Sampling Method

Implementing a FIR filter in DSP hardware such as the DSP56001 is a relatively simple task. Determining a set of coefficients that describe a given impulse response of a filter is also a straightforward procedure. However, deriving the optimal coefficients necessary to obtain a particular response in the frequency domain is not always as easy. The following paragraphs introduce a simple method to determine a set of coefficients based on a desired arbitrary frequency response (often referred to as the frequency sampling method). This method has the distinct advantage of being done in real time. However, the most efficient determination of coefficients is best done by utilizing a software filter design system such as FDAS to perform numerical curve-fitting and optimization, a procedure that necessitates using a computer. For example, the inverse Fourier transform integral is used to determine the FIR coefficients from a response specification in the frequency domain, which generally requires a numerical integration procedure. The inverse discrete Fourier transform (IDFT), which can be implemented using the inverse fast Fourier transform (IFFT) algorithm, is discussed in the following paragraphs. One reason for choosing this approach is to demonstrate a method that can be used to determine coefficients in real time since the fast Fourier transform (FFT) can be implemented in the same DSP56001 hardware as the FIR filter.

The question is "How must the filter be specified in the frequency domain so that Eqn. (B.44) and Eqn. (B.45) are realized?". That is, starting with the definition of the filter in the frequency domain, a method for calculating the filter coefficients is required. Beginning with Eqn. (B.36), the z -transform of the FIR filter, and evaluating this transfer function on the unit circle at N equally spaced normalized frequencies (i.e., $z = e^{j2\pi k/N}$) produces:

$$H(k) = H(e^{j2\pi k/N}) = \sum_{i=0}^{N-1} h(i) e^{-j2\pi ki/N} \quad (\text{B.46})$$

where: $0 \leq k \leq N-1$

$h(i)$ can be solved in terms of the frequency response at the discrete frequencies, $H(k)$, by multiplying both sides of Eqn. (B.46) by $e^{j2\pi km/N}$ and summing over k as follows:

$$\begin{aligned}
 \sum_{k=0}^{N-1} H(k) e^{j2\pi km/N} &= \sum_{k=0}^{N-1} \sum_{i=0}^{N-1} h(i) e^{-j2\pi ki/N} e^{j2\pi km/N} \\
 &= \sum_{i=0}^{N-1} h(i) \sum_{k=0}^{N-1} e^{-j2\pi ki/N} e^{j2\pi km/N} \\
 &= \sum_{i=0}^{N-1} h(i) N \delta_{im} \\
 &= N h(m)
 \end{aligned} \tag{B.47}$$

where: δ_{im} is the Kronecker delta, which is equal to one when $i = m$ but is zero otherwise

Eqn. (B.47) can be used to find $h(i)$ by simply setting $i = m$.

$$h(i) = \frac{1}{N} \sum_{k=0}^{N-1} H(k) e^{j2\pi ki/N} \tag{B.48}$$

Eqn. (B.48) is the IDFT of the filter response. In general, the discrete Fourier coefficients are complex; therefore, $H(k)$ can be represented as:

$$H(k) = A(k) e^{j\phi(k)} \tag{B.49}$$

where: $A(k) \equiv |H(k)|$

so that Eqn. (B.48) becomes:

$$h(i) = \frac{1}{N} \sum_{k=0}^{N-1} A(k) e^{j\phi(k)} e^{j2\pi ki/N} \tag{B.50}$$

At this point, the linear-phase constraints, Eqn. (B.44) and Eqn. (B.45), can be applied. The constraint that the coefficients be real and symmetrical when $h(i)$ is complex (see Reference 1) implies that:

$$h(i) = h^*(N-1-i) \tag{B.51}$$

where: * signifies the complex conjugate

Therefore:

$$\begin{aligned}
 h^*(N-1-i) &= \frac{1}{N} \sum_{k=0}^{N-1} A(k) e^{-j\phi(k)} e^{-j2\pi k(N-1-i)/N} \\
 &= \frac{1}{N} \sum_{k=0}^{N-1} A(k) e^{-j[\phi(k) + 2\pi k(N-1)/N]} e^{j2\pi ki/N}
 \end{aligned} \tag{B.52}$$

Eqn. (B.52) will be identical to Eqn. (B.50) if:

$$\phi(k) = -\phi(k) - \frac{2\pi k(N-1)}{N} + 2\pi r \quad \text{for } r=0,1,2,\dots$$

or

$$\phi(k) = \pi r - \pi k \frac{(N-1)}{N} \tag{B.53}$$

What are the constraints on r and $A(k)$ which will guarantee a purely real response for N even?

Substituting Eqn. (B.53) into Eqn. (B.50) yields:

$$h(i) = \frac{1}{N} \sum_{k=0}^{N-1} A(k) e^{j[\pi r - \pi k(N-1)/N + 2\pi ki/N]} \tag{B.54}$$

Expanding Eqn. (B.54) for even N yields:

$$\begin{aligned}
 h(i) &= \frac{1}{N} \left\{ A(0) e^{j\pi r} + A(1) e^{j[\pi r - \pi(N-1)/N + 2\pi i/N]} + \dots \right. \\
 &\quad \left. + A(N/2) e^{j[\pi r - \pi(N-1)/2 + \pi i]} + \dots \right. \\
 &\quad \left. + A(N-1) e^{j[\pi r - \pi(N-1)(N-1)/N + 2\pi(N-1)i/N]} \right\}
 \end{aligned}$$

Consider the $A(k)$ and $A(N-k)$ terms; if $r = 0$ for the $A(k)$ term and $r = 1$ for the $A(N-k)$ term, then the argument for the $A(N-k)$ term is the negative of the argument for the $A(k)$ term, given that N is even. That is:

$$\begin{aligned}
 &e^{j[\pi - \pi(N-k)(N-1)/N + 2\pi(N-k)i/N]} \\
 &= e^{j[\pi k(N-1)/N - 2\pi ki/N]} e^{j[\pi - \pi(N-1) + 2\pi i]} \\
 &= e^{-j[-\pi k(N-1)/N + 2\pi ki/N]} e^{-j[\pi(N-2) + 2\pi i]} \\
 &= e^{-j[-\pi k(N-1)/N - 2\pi ki/N]}
 \end{aligned}$$

Therefore, if $A(k) = A(N-k)$, then all imaginary components in Eqn. (B.54) will cancel and $h(i)$ will be purely real, which is the desired result. In general, for N even, the formulas for the frequency sampling method can be reduced to:

$$\begin{aligned}
 h(i) &= \frac{1}{N} \left\{ A(0) + \sum_{k=1}^{N-1} A(k) e^{j[\pi r - \pi k(N-1)/N + 2\pi ki/N]} \right\} \\
 &= \frac{1}{N} \left\{ A(0) + \sum_{k=1}^{N/2-1} A(k) [e^{j[\pi r - \pi k(N-1)/N + 2\pi ki/N]} + \right. \\
 &\quad \left. e^{j[\pi - \pi(N-k)(N-1)/N + 2\pi(N-k)i/N]} \right\} \\
 &= \frac{1}{N} \left\{ A(0) + \sum_{k=1}^{N/2-1} A(k) [e^{-j\pi k(N-1)/N + 2\pi ki/N} + \right. \\
 &\quad \left. e^{-j[\pi k(N-1)/N + 2\pi ki/N]} \right\} \\
 &= \frac{1}{N} \left\{ A(0) + \sum_{k=1}^{N/2-1} A(k) [e^{-j\pi k} e^{j[\pi k/N + 2\pi ki/N]} + \right. \\
 &\quad \left. e^{j\pi k} e^{-j[\pi k/N + 2\pi ki/N]} \right\} \\
 &= \frac{1}{N} \left\{ A(0) + 2 \sum_{k=1}^{N/2-1} A(k) (-1)^k \cos \frac{\pi k}{N} (1 + 2i) \right\} \tag{B.55}
 \end{aligned}$$

where: $r=0$ for $0 \leq k < N/2$

$r=1$ for $N/2 < k \leq N-1$

given

$$A(k) = A(N-k) \quad \text{for } 1 \leq k \leq \frac{N}{2} - 1$$

with

$$A(N/2) = 0$$

and

$$\phi(k) = -\pi k \frac{N-1}{N} \quad \text{for } 0 \leq k \leq \frac{N}{2} - 1$$

$$\phi(k) = \pi - \pi k \frac{N-1}{N} \quad \text{for } \frac{N}{2} + 1 \leq k \leq N-1$$

In summary, if a filter with an even number of symmetrical real coefficients is desired, then the phase must be linear, and the frequency response must be symmetrical about $N/2$ and zero at $N/2$ (see Reference 1).

As an example, consider the arbitrary filter specified in Figure B.48. In this example, $N = 32$ and an arbitrary lowpass and bandpass combination is specified. Figure B.49 shows the result of transforming the polar coordinate filter specification into

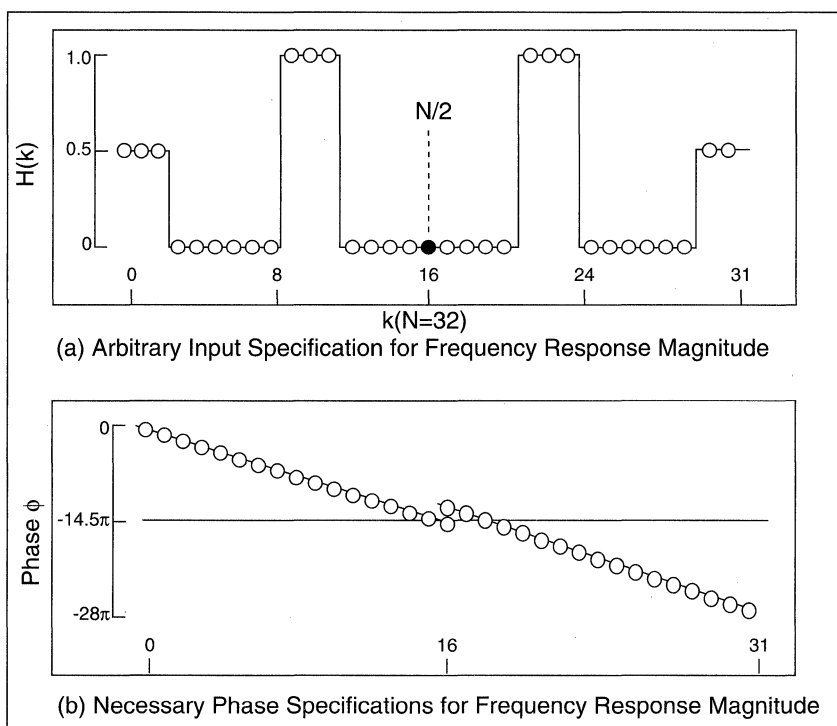


Fig. B.48 Arbitrary Filter Example

rectangular coordinates, yielding real and imaginary components of the transfer function. This transformation is accomplished by treating the magnitude of $H(k)$ as the length of a vector in polar coordinates and the phase as the angle. The x component (real part) is the product of the length (magnitude of $H(k)$) and the cosine of the phase. Likewise, the y component (imaginary part) is the product of the length and the sine of the phase angle. This transformation to rectangular coordinates is necessary to perform the IDFT (or IFFT) calculation:

$$h(n) = \frac{1}{N} \sum_{k=0}^{N-1} H(k) e^{jk\theta n} \quad (\text{B.56})$$

where: $H(k)$ is a complex number and $\theta n = 2\pi n/N$

When the filter's transfer function, $H(k)$, is described as in Figure B.48 and Figure B.49 (i.e., the real part symmetric about $N/2$ and the imaginary part asymmetric about $N/2$), $h(n)$ is strictly real. If $h(n)$ were complex, the FIR filter would be much more difficult to implement since twice as many terms would be present. Figure B.50 shows the results of the IDFT applied to the example arbitrary filter specification.

Note the symmetry of the coefficients in Figure B.50 (symmetric about $(N-1)/2$). This symmetry is to be expected for an even number of coefficients. Eqn. (B.57) has

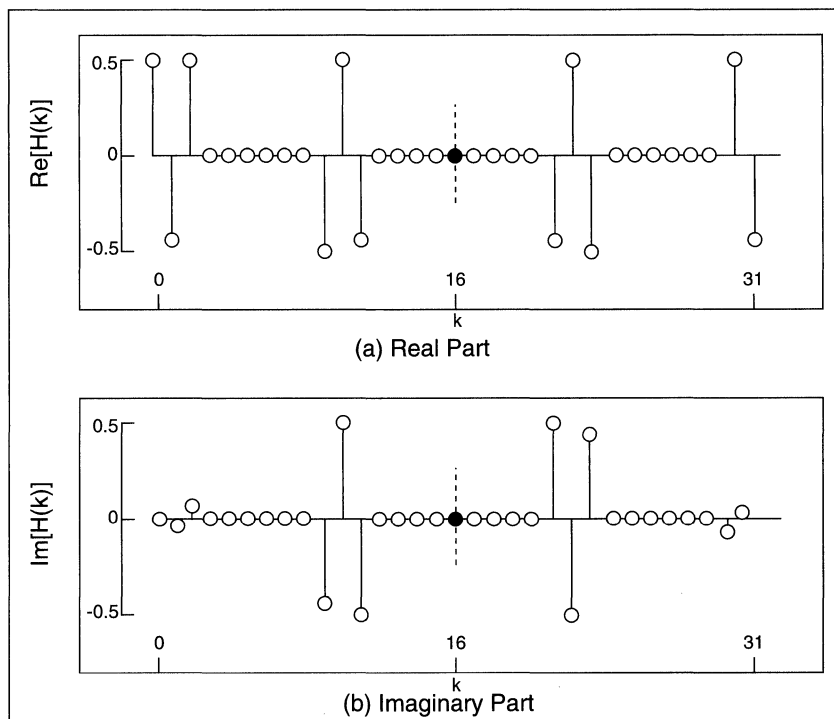


Fig. B.49 Response Transformed from Polar Coordinates

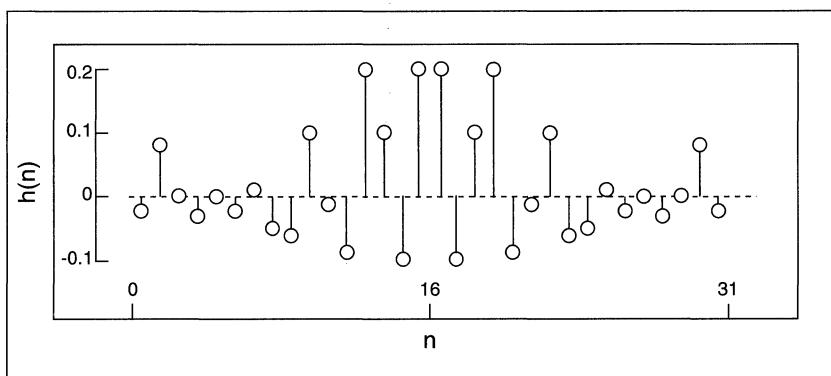


Fig. B.50 FIR Coefficients from Eqn. (B.36) for Filter Example

$N-1$ roots since it is a polynomial of order $N-1$. These roots are plotted in Figure B.51 and must be found from a numerical algorithm such as the Newton-Raphson root-finding recursion relation or the Mueller method (see Reference 18). $h(n)$ can now be used to specify the filter response in the continuous frequency domain by setting $\theta = 2\pi f/f_s$.

The transfer function has exactly the same form as the DFT of the $h(n)$, but now the frequency is continuous up to $f_s/2$:

$$H(\theta) = \sum_{n=0}^{N-1} h(n)e^{-jn\theta} \quad (\text{B.57})$$

where: $\theta = 2\pi f/f_s$

The continuous frequency gain and phase response of the 32-coefficient example filter, plotted in Figure B.52, are generated as follows:

$$\begin{aligned} G(\theta) &= |H(\theta)| = [H(\theta)H^*(\theta)]^{1/2} \\ &= \left\{ \left[\sum_{n=0}^{N-1} h(n)\cos n\theta \right]^2 + \left[\sum_{n=0}^{N-1} h(n)\sin n\theta \right]^2 \right\}^{1/2} \end{aligned} \quad (\text{B.58})$$

and

$$\phi(\theta) = (-\tan^{-1}) \left\{ \frac{\sum_{n=0}^{N-1} h(n)\sin n\theta}{\sum_{n=0}^{N-1} h(n)\cos n\theta} \right\} \quad (\text{B.59})$$

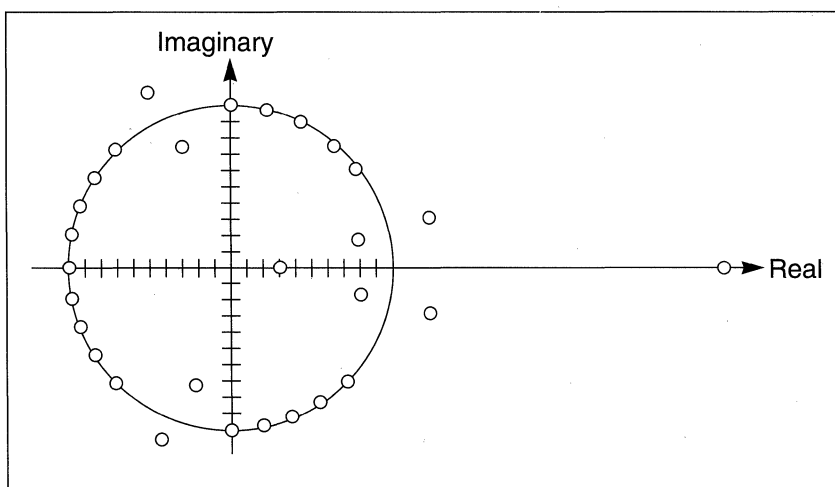


Fig. B.51 Roots (Zeros) of Eqn. (B.57) for Filter Example

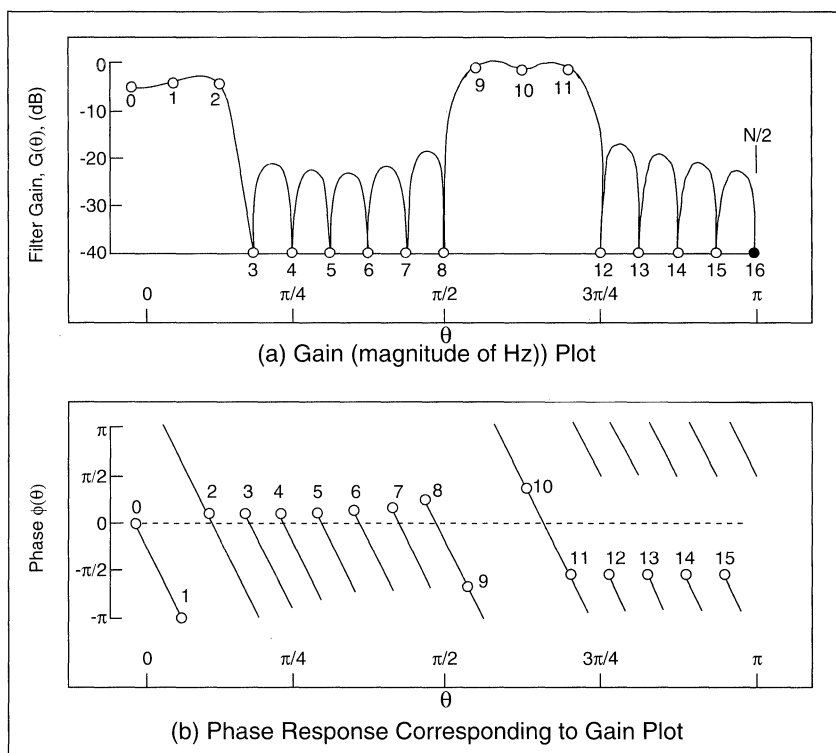


Fig. B.52 A 32-Point FIR Filter Example

where: $h(n)$ is the value obtained from Eqn. (B.56) or equivalently from Eqn. (B.55)

Note that the gain plot (see Figure B.52) exactly intersects the discrete frequency points originally specified in Figure B.48. Clearly, $G(\theta)$ and $\phi(\theta)$ have many discontinuities. Due to the symmetry of linear-phase FIR filters, analytic expressions can be found for $H(\omega)$ as shown in Eqn. (B.63) and Eqn. (B.64).

For comparison, Figure B.53 shows the log magnitude response of the example filter with larger values of N . The stopband attenuation is not improved as much as expected when the number of coefficients is increased; however, the sharpness of the transition band is significantly enhanced because the approach discussed has implicitly assumed a rectangular window function. A smoothing window, $w(n)$, can be used to improve this situation (see Reference 10).

If $h(n)$ is modified by a window function, $w(n)$, $h_w(n) = h(n)w(n)$ for $0 \leq n \leq N-1$, the gain of Eqn. (B.58) (shown in Figure B.54 (a)) can be greatly improved. The window function, $w(n)$, goes to zero at both ends ($n = 0$ and $n = N$) and is unity at the center; $w(n)$ is symmetrical so as not to disrupt the linear-phase characteristics of the filter. Figure B.54 (b) uses a window function described by $w(n) = \sin^2[\pi n/(N-1)]$ (also known as a Hanning window) to demonstrate the sensitivity of the gain to windowing. $w(n)$ is basically an envelope function used to taper the ends of $h(n)$ smoothly. The rounding of

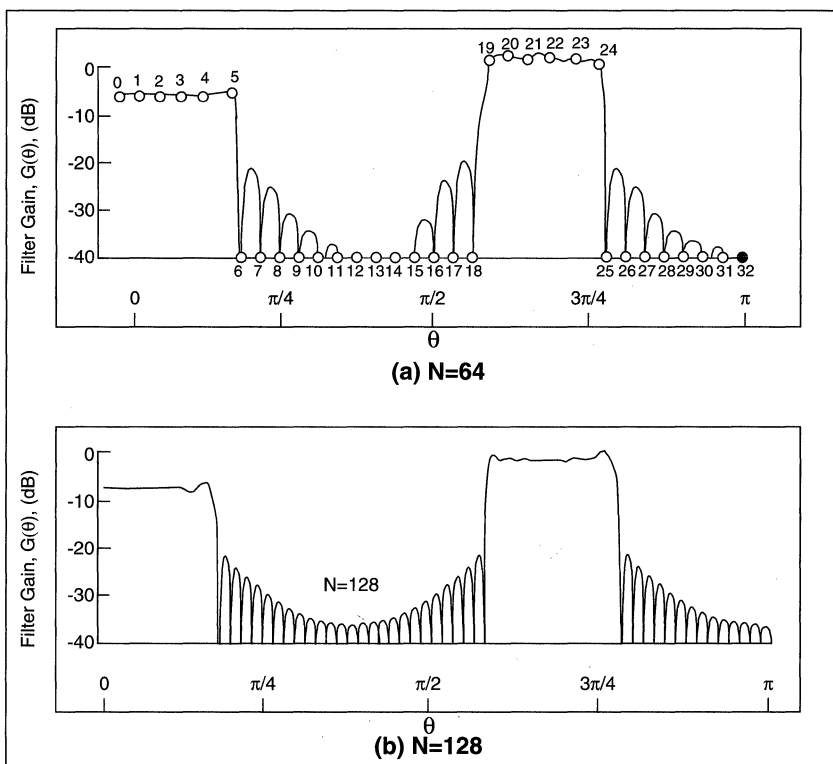


Fig. B.53 Log Magnitude Response of Filter Example with Larger N Values

the transition-band edges in Figure B.54 (b) is the trade-off for windowing; however, in most applications, this trade-off is well worthwhile.

Window functions have a powerful effect on the stopband attenuation as well as the passband transition slope. The best passband transition slope performance is achieved with the rectangular window, but this window results in very poor stopband performance and often severe passband fluctuation (or ripple). All window functions have the effect of increasing the stopband attenuation and reducing the passband ripple at the expense of increasing the width of the transition region. However, for most window functions, the passband ripple is relatively insensitive to N . Windowing is described in virtually any DSP textbook (see References). Also, windowing is discussed with practical examples in *Implementation of Fast Fourier Transforms on Motorola's DSP56000/DSP56001 and DSP96002 Digital Signal Processors* (see Reference 19).

B.7.3 FIR Filter Design Using FDAS

Figure B.55 shows an example (log magnitude and impulse response) of a band-pass filter generated with the FDAS software package using the Kaiser window. A

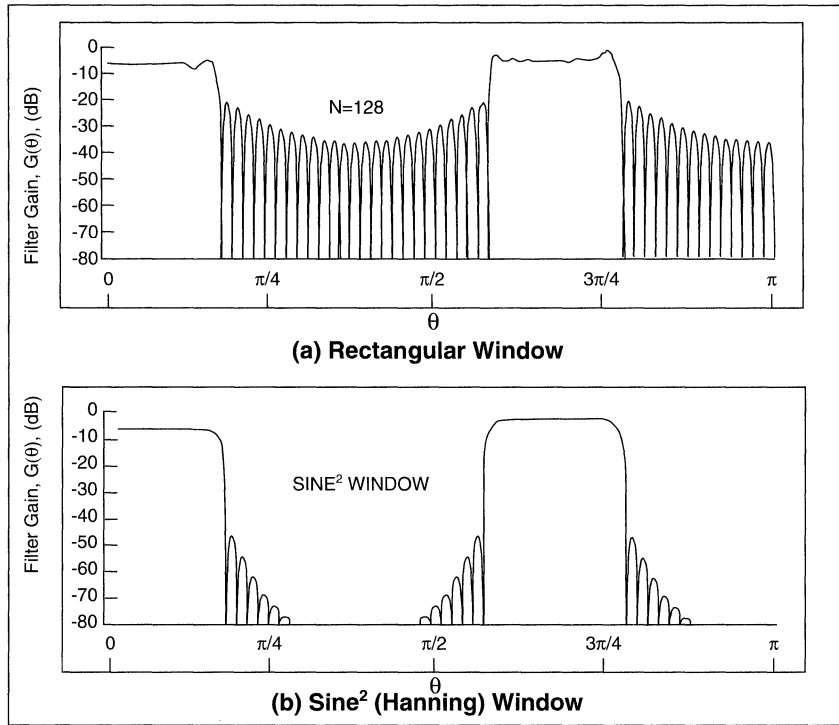


Fig. B.54 Window Function Effects on Filter Example

totally different approach to FIR filter design, the equiripple method, is based on finding an optimum approximation to the ideal or desired response, $D(\theta)$. An optimum approximation can be found because of the inherent symmetry of the coefficients of linear-phase filters.

Recall that the continuous frequency response of a FIR filter can be found by setting $z = e^{j\theta}$ in the z -transform so that:

$$H(\theta) = \sum_{n=0}^{N-1} h(n)e^{-j\theta n} \quad (\text{B.60})$$

If the linear-phase factor is factored out, Eqn. (B.60) can be written as:

$$\begin{aligned} H(\theta) &= e^{-j\theta(N-1)/2} \sum_{n=0}^{N-1} h(n)e^{j\theta[(N-1)/2-n]} \\ &= e^{-j\theta(N-1)/2} \{h(0)e^{j\theta(N-1)/2} + h(1)e^{j\theta[(N-1)/2-1]} + \dots \\ &\quad h(N-1)e^{-j\theta(N-1)/2} + h(N-2)e^{-j\theta[(N-1)/2-1]}\} \end{aligned} \quad (\text{B.61})$$

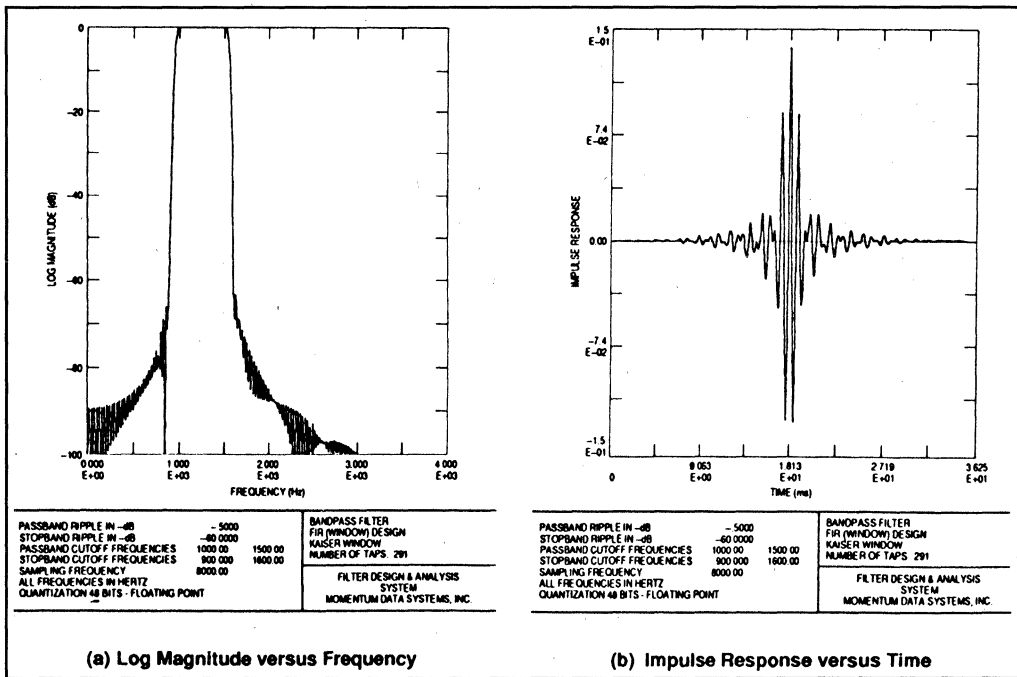


Fig. B.55 FDAS Output for FIR Bandpass Filter Example with a Kaiser Window

Using Euler's identity, Eqn. (B.61) can be grouped into sine and cosine terms as follows:

$$\begin{aligned}
 H(\theta) = e^{-j\theta(N-1)/2} & \{ [h(0) + h(N-1)] \cos \left[\theta \frac{(N-1)}{2} \right] \\
 & + jh0 - hN - 1 \sin \left[\theta \frac{N-1}{2} \right] + [h(1) + h(N-2)] \cos \left[\theta \frac{(N-1)}{2} - 1 \right] \\
 & + j[h(1) - h(N-2)] \sin \left[\theta \frac{N-1}{2} - 1 \right] \}
 \end{aligned} \quad (B.62)$$

If it is assumed that linear phase is achieved by even symmetry (i.e., $h(n) = h(N-1-n)$) and that N is even, Eqn. (B.62) reduces to:

$$H(\theta) = e^{-j\theta(N-1)/2} \sum_{n=0}^{N/2-1} 2h(n) \cos \left[\theta \left(\frac{N-2}{2} - n \right) \right]$$

which, with a change of variable, $k = N/2 - n - 1$, can be written as follows:

$$H(\theta) = e^{-j\theta(N-1)/2} \sum_{n=0}^{N/2-1} 2h(n) \cos \left[\theta \left(\frac{N-2}{2} - n \right) \right] \quad (B.63)$$

which is in the following form:

$$H(\theta) = A(\theta)e^{jP\theta} \quad (\text{B.64})$$

where: $A(\theta)$ is a real value amplitude function

$P(\theta)$ is a linear-phase function

The linear functions, $A(\theta)$ and $P(\theta)$, are to be contrasted with the inherently non-linear absolute value and arctangent functions in Eqn. (B.58) and Eqn. (B.59). For all four types of linear-phase filters (N even or odd and symmetry even or odd), $A(\theta)$ can be expressed as a sum of cosines (see Reference 21).

Since an ideal filter cannot be realized, an approximation must be used. If the desired frequency response, $D(\theta)$, can be specified in terms of a deviation, δ , from the ideal response, then the error function in Eqn. (B.65) can be minimized:

$$\|E(\theta)\| = \max_{\theta} |D(\theta) - A(\theta)| \quad (\text{B.65})$$

by finding the best $A(\theta)$. Because $A(\theta)$ can be expressed as a finite sum of cosines as shown in Eqn. (B.63), it can be shown (see Reference 18) that the optimal $A(\theta)$ will be unique and will have at least $N/2 + 1$ extremal frequencies, where extremal frequencies are points such that for:

$$\theta_1 < \theta_2 < \dots < \theta_{N/2} < \theta_{N/2+1}$$

$$E(\theta_e) = -E(\theta_{e+1}) \quad \text{for } e = 1, 2, \dots, \frac{N}{2} + 1$$

and

$$|E(\theta_e)| = \max_{\theta} [E(\theta)]$$

Thus, the best approximation will exhibit an equiripple error function. The problem reduces to finding the extremal frequencies since, once they are found, the coefficients, $2h(N/2-k-1)$, can be found by solving the set of linear equations:

$$D(\theta) \pm \delta = A(\theta_e) = \sum_{k=0}^{N/2-1} 2h\left(\frac{N}{2} - k - 1\right) \cos\left[\theta_e\left(k + \frac{1}{2}\right)\right] \quad (\text{B.66})$$

The Remez exchange algorithm is used to systematically find the extremal frequencies (see Reference 5). Basically, a guess is made for the initial $N/2 + 1$ extremal frequencies. (Usually, this guess consists of $N/2 + 1$ equally spaced frequencies in the Nyquist range.) Using this guess, Eqn. (B.66) is solved for the coefficients and δ . Using these coefficients, $A(\theta)$ is calculated for all frequencies and the extrema, and frequencies at which the extrema are attained are determined. If the extrema are all equal and equal to or less than that specified in the initial filter specification, the problem is solved. However, if this is not the case, the frequencies at which the extrema were attained are used as the next guess. Note that the final extremal frequencies do not have to be equally spaced. Clearly, the equiripple design approach is calculation intensive.

What is the benefit of equiripple designs over window designs? In general, equiripple designs require fewer taps for straightforward requirements. When the specification requires a sharp cutoff and/or a large stopband attenuation or a narrow bandpass, the equiripple approach may fail to converge. In general, when N is decreased, an equiripple design tends to maintain its transition band while sacrificing stopband attenuation; window designs tend to do the opposite. Of the window alternatives, the Kaiser window is preferred for designing filters because the passband ripple and stopband attenuation can be varied relatively independent of the transition width (see Reference 1). For spectral analysis, the Blackman-Harris window is preferred (see Reference 10).

Figure B.56 shows an example of an equiripple design generated from the FDAS software package. This example is the same as that used for the Kaiser window example of Figure B.55. The number of coefficients in the equiripple design is far less than that generated by the Kaiser window method (179 versus 291). However, the passband ripple is larger.

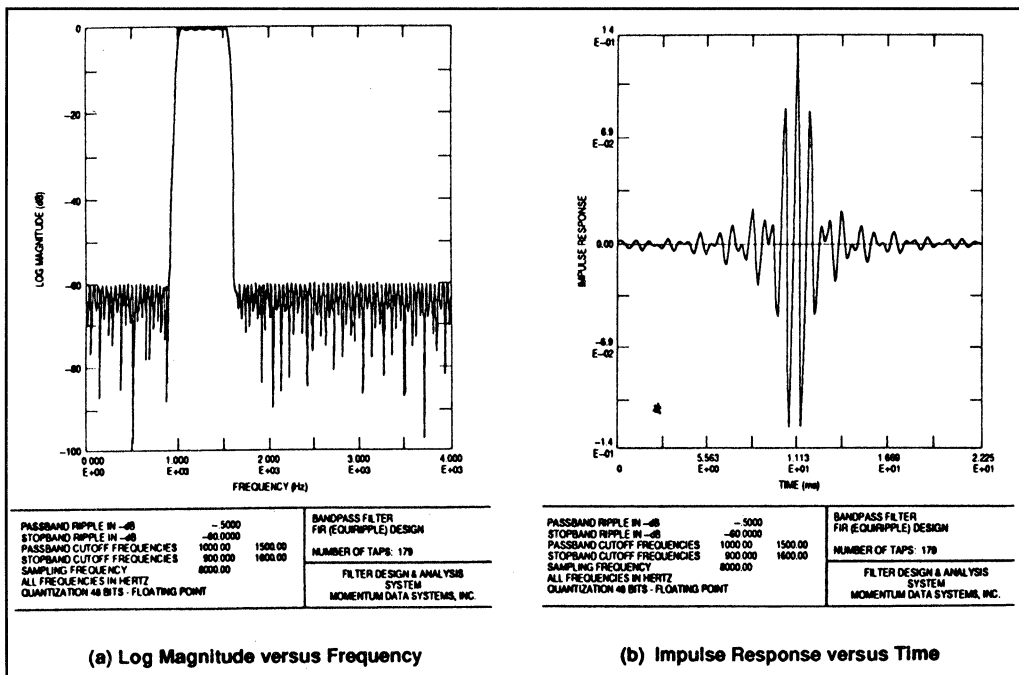


Fig. B.56 FDAS Output for FIR Bandpass Filter Example with Equiripple Design

B.7.4 FIR Implementation on the DSP56001

The DSP56001 has several architectural features that make it ideally suited for implementing FIR filters:

1. Dual Harvard architecture uses two data memories with dedicated buses and address generation units, allowing two addresses to be generated in a single cycle. If one address is pointing to data and another address is pointing to coefficients, a word of data and a coefficient can be fetched in a single cycle.
2. Modulo addressing makes the shifting of data unnecessary. If an address pointer is incremented (or decremented) with the modulo modifier in effect, data shifting can be accomplished by just “backing up” the address register by one to overwrite the data that would normally be shifted out. This procedure allows very efficient addressing of operands without wasting time shifting the data or reinitializing pointers.
3. Hardware DO loops execute without overhead once the loop is started. After a three-cycle initialization of the DO loop, the body of the loop executes as if it were straight-line code. Since the DO loop does not require any overhead cycles for each pass, the need for straight-line code is eliminated.

For a four-coefficient example of a linear-phase FIR filter, the input-output difference equation can be found by expanding Eqn. (B.35):

$$y(n) = h_0x(n) + h_1x(n-1) + h_2x(n-2) + h_3x(n-3) \quad (\text{B.67})$$

This difference equation can be realized with the discrete-time four-tap filter example shown in Figure B.57. The filter can be efficiently implemented on the DSP56001 by using modulo addressing to implement the shifting and parallel data moves to load the multiplier-accumulator. The filter network is shown in Figure B.57 (a); the memory map for the filter inputs and coefficients is shown in Figure B.57 (b). The following DSP56001 code is used to implement the direct form FIR filter:

```
CLR    A          XO,X:(R0)+   Y:(R4)+,Y0   ;Save input sample, fetch coef.
REP    #NTAPS-1    ;Repeat next instruction.
MAC    XO,Y0, A    X:(R0)+,X 0   Y:(R4)+,Y0   ;FIR Filters.
MACR   XO,Y0, A    (R0)-        ;Round result and adjust R0.
```

Register R0 points to the input variable buffer, M0 is set to three (modulo 4), R4 points to the coefficient buffer, and M4 is set to three. The input sample is in X0.

The CLR instruction clears accumulator A and performs parallel data moves. The data move saves the most recent input value to the filter (assumed to be in X0) into the location occupied by the oldest data in the shift register and moves the first coefficient in the filter (h_0) into the data ALU.

The REP instruction repeats the next instruction NTAPS-1 times. Since there are four taps in this filter, the next instruction is repeated three times.

The MAC instruction multiplies the data in X0 by the coefficient in Y0 and adds the result to accumulator A in a single cycle. The data move in this instruction loads the next input data variable into X0 and the next coefficient into Y0. Both address registers R0 and R4 are incremented. The MACR instruction calculates the final tap of the filter, rounds the result using convergent rounding, and address register R0 is decremented.

Address register R4 is incremented once before the REP instruction and three times due to the REP instruction for a total of four increments. Since the modulus for R4 is four, the value of R4 wraps around pointing back to the first coefficient.

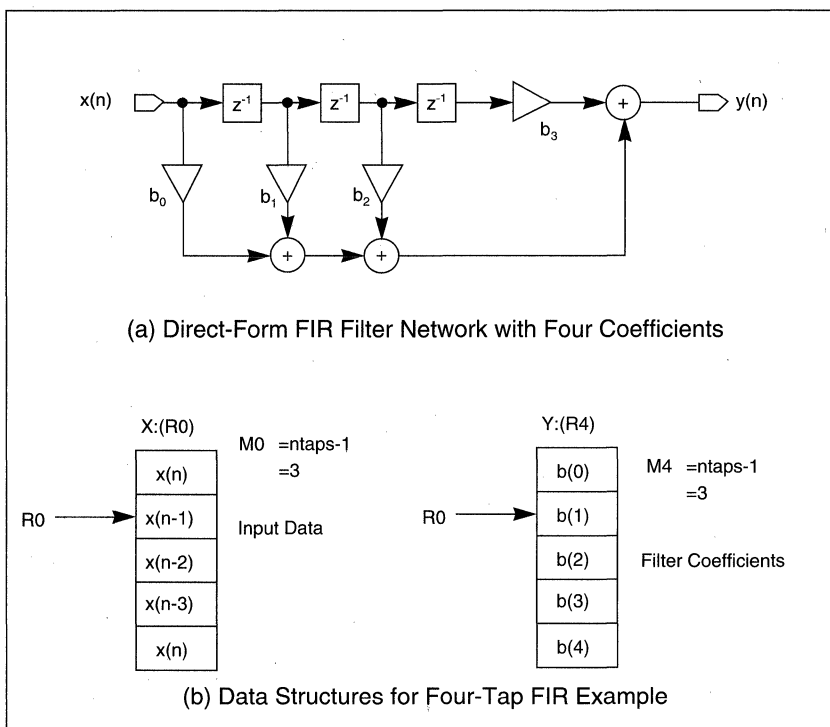


Fig. B.57 FIR Filter Example

The operation of $R0$ is similar. The value input to the filter is saved, and then $R0$ is incremented pointing to the first state. The REP instruction increments $R0$ three times. Since the modulo on $R0$ is four and $R0$ is incremented four times, the value in $R0$ wraps around pointing to the new input sample. When the $MACR$ instruction is executed, the value of $R0$ is decremented, pointing to the old value, $x(n-3)$. The next sample time overwrites the value of $x(n-3)$ with the new input sample, $x(n)$. Thus, the shifting of the input data is accomplished by simply adjusting the address pointer, and the modular addressing wraps the pointer around at the ends of the input data buffer. Instruction cycle counts for this filter are $NTAPS+3$. Thus, for a four-tap filter, seven instructions are required.

B.8 REFERENCES

1. Antoniou, Andreas, *Digital Filters: Analysis and Design*, New York, NY: McGraw-Hill, 1974.
2. Berlin, Howard M., *Design of Active Filters with Experiments*, Indianapolis, IN: Howard W. Sams & Co., Inc., 1977.

3. Bogner and Constantinides, eds., *Introduction to Digital Filtering*, New York, NY: John Wiley & Sons, 1975.
4. Brophy, J. J., *Basic Electronics for Scientists*, New York, NY: McGraw-Hill, 1966.
5. Cheney, E. W., *Introduction to Approximation Theory*, New York, NY: McGraw-Hill, 1966.
6. Chrysafis, A., *Fractional and Integer Arithmetic Using the DSP56000 Family of General-Purpose Digital Signal Processors* (APR3/D), Motorola, Inc., 1988.
7. "Digital Filters on DSP56000/DSP56001," Motorola Technical Bulletin, 1988.
8. *Digital Stereo 10-Band Graphic Equalizer Using the DSP56001* (APR2/D), Motorola, Inc., 1988.
9. "Filter Design & Analysis System," Version 1.3, Momentum Data Systems, 1985.
10. Harris, Fredric J., "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proc. of IEEE*, vol. 66, no. 1, 1978, pp.51-83.
11. Jackson, Leland B., *Digital Filters and Signal Processing*, Boston, MA: Kluwer Academic Publishers, 1986.
12. Lancaster, D., *Active Filter Cookbook*, Indianapolis, IN: Howard W. Sams & Co., Inc., 1975.
13. McClellan, J. H. and T. W. Parks, "A Unified Approach to the Design of Optimum FIR Linear-Phase Digital Filters," *IEEE Trans. Circuits Systems CT-20*, 1973, pp. 697-701.
14. Moschytzm, G. S. and P. Horn, *Active Filter Design Handbook*, New York, NY: John Wiley & Sons, 1981.
15. Oppenheim, A. V. and R. W. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1975.
16. Parks, T. W. and J. H. McClellan, "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase," *IEEE Trans. Circuit Theory CT-19*, 1972, pp. 189-194.
17. Proakis, John G. et al., *Introduction to Digital Signal Processing*, New York, NY: Macmillan, 1988.
18. Rabiner, L. R. and B. Gold, *Theory and Application of Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1975.
19. Sohie, Guy R. L., *Implementation of Fast Fourier Transform on Motorola's DSP56000/DSP56001 and DSP96002 Digital Signal Processors* (APR4/D), Motorola, Inc., 1989.
20. Strawn, J., et al., *Digital Audio Signal Processing—An Anthology*, William Kaufmann, 1985.
21. Williams, Arthur B., *Electronic Filter Design Handbook*, New York, NY: McGraw-Hill, 1981.

Implementation of Fast Fourier Transforms on Motorola's Digital Signal Processors

Please note that the DSP56001 is obsolete. It has been replaced by the DSP56001A/56002, depending on degree of compatibility required.

C.1 INTRODUCTION TO THE FOURIER INTEGRAL

"... a digital signal processor can efficiently compute the Fourier transform and perform specific frequency-domain tasks. . ."

C.1.1 Definition and History

The scientific and engineering communities have attempted to represent changing signals in two fundamental domains: time and frequency. Temporal changes are easily shown on oscilloscopes, for instance, where change in time is directly proportional to distance across a screen. Representation of signals in terms of frequencies falls under the general category of "spectrum analysis", and has generated a lot of attention recently, due to the increased availability of hardware which makes such representations possible. The first formal approach to spectrum analysis probably dates back to the work of Fourier, who showed how to mathematically represent a general class of time-varying phenomena in terms of sine and cosine functions of particular frequencies. His work is best known as the *Fourier Integral* (inverse Fourier transform) (see Reference 1):

$$\chi(t) = \int_{-\infty}^{+\infty} X(f)e^{j2\pi ft}df \quad (C.1)$$

where: $j = \sqrt{-1}$ and $e^{j2\pi ft} = \cos(2\pi ft) + j\sin(2\pi ft)$

When interpreted as an infinite summation, the previous integral is simply a linear combination of a number of sine and cosine functions (expressed by the complex exponential), each one of which is weighted by the complex amplitude $X(f)$. Conversely, the complex frequency function $X(f)$ can be derived from the time-varying signal $\chi(t)$ by the *Fourier Transform*:

$$X(f) = \int_{-\infty}^{+\infty} \chi(t)e^{-j2\pi ft} dt \quad (C.2)$$

The two expressions shown in Eqn. (C.1) and Eqn. (C.2) define a Fourier transform pair $\chi(t)$ and $X(f)$. The Fourier transform $X(f)$ determines the frequency content of the signal in question, while $\chi(t)$ shows the way the signal varies as a function of time. Note that, in general, $\chi(t)$ can be directly measured (for instance, displayed on an oscilloscope). $X(f)$ remains a mathematical expression which attempts to express our intuitive perception of frequency.

Unfortunately, it is not always true that the concept of frequency, as defined by the Fourier transform in Eqn. (C.2), and the intuitive concept of frequency as we perceive it, are identical. For instance, music consists of tones (frequencies) which vary over time. Although we can clearly perceive time-varying frequencies, Eqn. (C.2) does not allow for Fourier's concept of frequency to have any time-varying character— $X(f)$ is a function of frequency only.

C.1.2 Use of the Fourier Transform

Because of the basic nature of the frequency concept, practical applications of the Fourier transform are abundant. As more cost-efficient methods become available to compute the Fourier transform, the number of practical solutions to frequency-based problems will grow even larger. In these frequency-based applications, a digital signal processor can efficiently compute the Fourier transform (as defined in **SECTION C.1.1 Definition and History**), and perform specific frequency-domain tasks such as elimination of certain frequency components, etc.

Three general types of Fourier transform applications are:

1. *Number-Based*—Most spectrum analysis applications require the direct evaluation of the Fourier transform as in Eqn. (C.2). Since the Fourier transform is a mathematical expression, these applications are based on numerical computations, and can be termed number-based. Examples range from spectrum analysis laboratory instrumentation and professional audio equipment to velocity estimation in radar. Note that in number-based applications the accuracy of the computed numbers is of vital importance to the performance of the overall system. For instance, the quality-conscious audio industry requires 16-bit or more precision in order to eliminate audible distortion.
2. *Pattern-Based*—Many problems involve the recognition and detection of signals with a specific frequency content (a predefined spectral pattern). For instance,

speech consists of segments of sound with very specific frequency characteristics. In this type of application, the conversion to the frequency domain is often only a single step in the overall task. It is important that this conversion process be as fast as practical, to allow for sufficient time to perform computationally intensive pattern matching techniques. In addition to providing fast Fourier transform computations, the processor in question needs to be fast at general-purpose DSP tasks so that it can perform a variety of frequency-based calculations for pattern matching.

3. *Convolution-Based*—The third class of applications of Fourier transforms uses the transform as a simple mathematical tool to perform general filtering in a very efficient manner. This concept is based on the property that the Fourier transform of the convolution of two time-signals:

$$y(t) = \int_{-\infty}^{+\infty} x(t-\tau)h(\tau)d\tau \quad (\text{C.3})$$

is equal to the product of the individual transforms:

$$Y(f) = X(f)H(f) \quad (\text{C.4})$$

Eqn. (C.3) (better known as the *convolution integral*) represents the output of a linear filter with *impulse response* $h(t)$ and input signal $x(t)$. Clearly, in the frequency domain, the output of a filter can be obtained by a simple multiplication, whereas in the time domain, a more complicated convolution integral needs to be solved. The amount of computation involved in evaluating the integral in Eqn. (C.3) becomes particularly large when the impulse response $h(t)$ has a long time duration which often prevents real-time implementation. Clearly, if the Fourier transform $X(f)$ of the signal can be computed efficiently, the filtering operation itself can be achieved by simple multiplications.

The combined number of computations (for computing the Fourier transform, for filtering in the frequency domain, and for obtaining the inverse Fourier Transform) is often less than the total number of calculations required to compute Eqn. (C.3) directly. This is especially true when the filter in question performs a simple frequency discrimination function (lowpass, bandpass, highpass, bandreject, etc.). In this case, the multiplications in the frequency domain can be replaced by a simple masking operation, which removes the stopbands and leaves the passband(s) unchanged.

Although no direct frequency information is extracted from the signal, the Fourier transform is used as a mathematical tool for fast-filtering applications. Note that again, fast Fourier transform and inverse Fourier transform “engines” are needed in order to provide the real-time filtering operation.

In summary, the basic nature of the frequency concept indicates that the number of possible frequency domain applications is as large as more conventional time domain applications. In the past, frequency domain applications were either difficult to implement or could not be realized in a cost-efficient manner because of the lack of

low-cost, high-performance hardware. This application note demonstrates that the DSP56001/2 and the DSP96002 Families of digital signal processors fulfill the demanding requirements imposed by frequency domain problems. In addition to providing a *fast* implementation of *high-precision* Fourier transform computations, the general-purpose nature of the instruction set allows for a **complete, single-chip, low-cost** integrated solution to a wide variety of frequency domain problems.

C.2 THE DISCRETE FOURIER TRANSFORM

"... the results need to be available within a finite time period, and the infinite summation must somehow be reduced to a finite summation."

C.2.1 The Discrete-Time Fourier Transform (DTFT)

In order to compute the Fourier transform using digital hardware, Eqn. (C.2) needs to be approximated in a manner which makes machine computation feasible. The first step in this process consists of eliminating the theoretical integral symbol, and replacing it by a computable sum:

$$X(f) \approx \tilde{X}(f) = T \sum_{n=-\infty}^{+\infty} \chi(nT) e^{-j2\pi f n T} \quad (C.5)$$

The above expression uses a sampled signal $\chi(nT)$, where the sampling period T is made as small as possible to reduce approximation errors. Appropriately, $\tilde{X}(f)$ is called the discrete-time Fourier transform (DTFT). As T (the sampling period) becomes infinitely small, the previous summation approaches the original Fourier transform in Eqn. (C.2). To assess the accuracy of this approximation, note that the resulting expression for $\tilde{X}(f)$ is a periodic function of frequency:

$$\tilde{X}(f) = \tilde{X}\left(f + \frac{1}{T}\right) \quad (C.6)$$

because:

$$e^{-j(2\pi f n T + j2\pi n \frac{T}{T})} = e^{-j2\pi f n T} e^{-j2\pi n} = e^{-j2\pi f n T} \quad (C.7)$$

In general, the original spectrum $X(f)$ is not periodic, and the approximation is only justified for a range of small values of f . In Figure C.1, the DTFT magnitude and the Fourier transform magnitude of a simple rectangular function are shown for several values of the sample rate $f_s = 1/T$. Note the periodic nature of the resulting function, as well as the approximation errors due to the sampling process.

The Nyquist sampling theorem gives a well accepted criterion for the sampling rate. It states that a signal needs to be sampled faster than twice its highest frequency. In other words, if:

$$X(f) = 0 \quad (C.8)$$

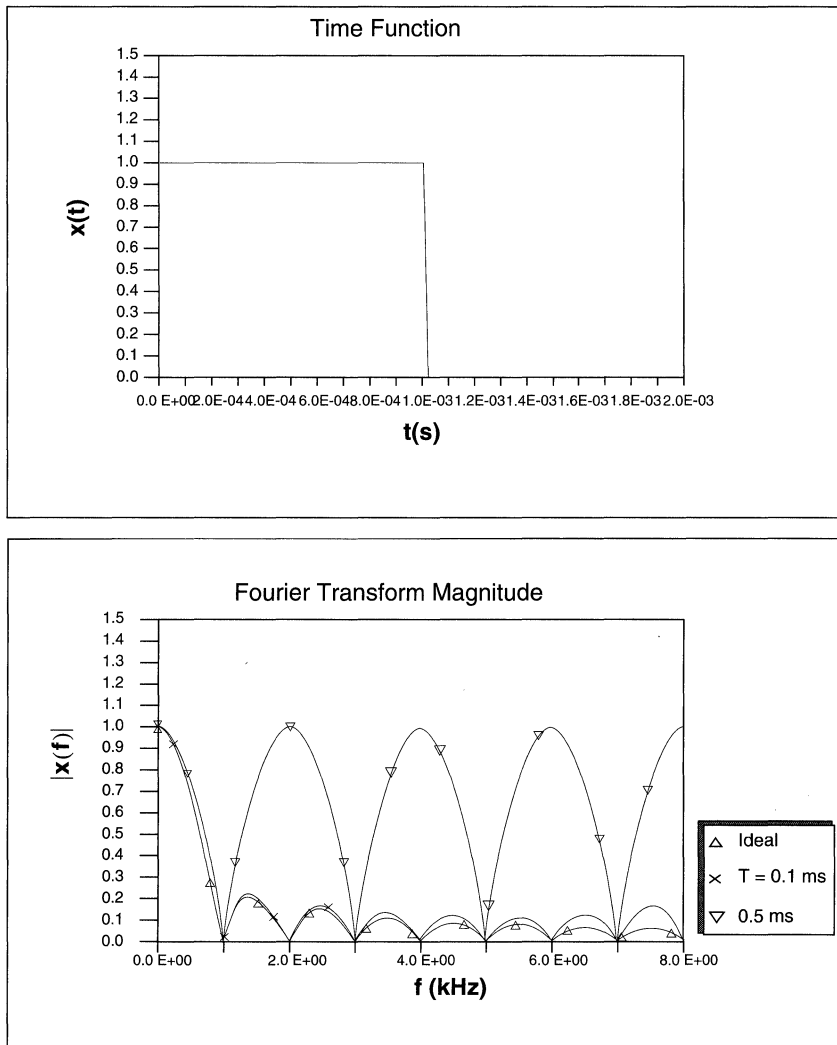


Fig. C.1 Fourier Transform of a Rectangular Function

for $|f| \geq B$ (B is referred to as the *bandwidth* of the signal), then the sampling frequency needs to satisfy:

$$f_s \geq 2B \quad (\text{C.9})$$

In practice, signals rarely satisfy Eqn. (C.9), and some error, called the *aliasing error*, can be expected in the evaluation of $X(f)$. The aliasing error is generated by frequency components at higher frequencies, which manifest themselves at lower frequencies because of the periodic nature of $\tilde{X}(f)$ (aliasing). The aliasing error can be

reduced by filtering out the higher-frequency components of the signal using a low-pass anti-aliasing filter and/or by increasing the sampling rate.

C.2.2 Windowing and Windowing Effects

The discussion of aliasing errors illustrates how the Fourier transform can be approximated by an infinite summation. In practice, the results need to be available within a finite time period, and the infinite summation must somehow be reduced to a finite summation. One obvious way to reduce the infinite summation is by simply truncating the sum in Eqn. (C.6) to N terms as:

$$\tilde{X}_w(f) = T \sum_{n=0}^{N-1} \chi(nT) e^{-j2\pi f n T} \quad (C.10)$$

This truncation is frequently referred to as “windowing” because an infinite summation is viewed through a finite window. The resulting transform is called the *windowed discrete-time* Fourier transform (WDFT). In mathematical terms, windowing is simply the multiplication of the signal by a window sequence of finite-length, $w(n)$. In the simple case above, $w(n)=1$ for $0 \leq n \leq N-1$; otherwise, $w(n)=0$. Because of its rectangular shape, the window shown above is called the *rectangular* window.

Unless the signal in question is of finite duration, this truncation will introduce other errors, resulting in a number of artifacts in the spectrum. To assess the effect of the windowing operation, a simple sine wave of the form:

$$\chi(t) = \sin(2\pi 1000t) \quad (C.11)$$

is sampled with a sampling frequency of 4000 Hz, and the windowed DTFT is computed with $N=20$.

Figure C.2 shows the result of windowing a sine wave by a rectangular window. Windowing causes the following errors:

1. *Leakage*—Even though the input signal consists of a single-frequency component at 1000 Hz, the result clearly shows components at frequencies other than 1000 Hz. This is called the leakage effect: it appears as if energy has “leaked” from 1000 Hz to the rest of the spectrum.
2. *Smoothing*—Although the theoretical transform exhibits an infinitely narrow, and infinitely large peak at 1000 Hz, the actual peak has finite magnitude and exhibits finite width. It appears that the narrow peak has been “smeared” out in the frequency domain as a result of the windowing function in the time domain. This effect is appropriately termed the smoothing effect.
3. *Ripple*—The overall magnitude plot in Figure C.2 shows an oscillatory character not present in the original Fourier transform: this is called the ripple effect. The origin of the ripple effect lies in the discontinuity (abrupt start and end) introduced in the signal by the window. Windows with more gradual transitions generally have lower sidelobes and less ripple.

In general, a tradeoff exists between these different effects, and the advantages of

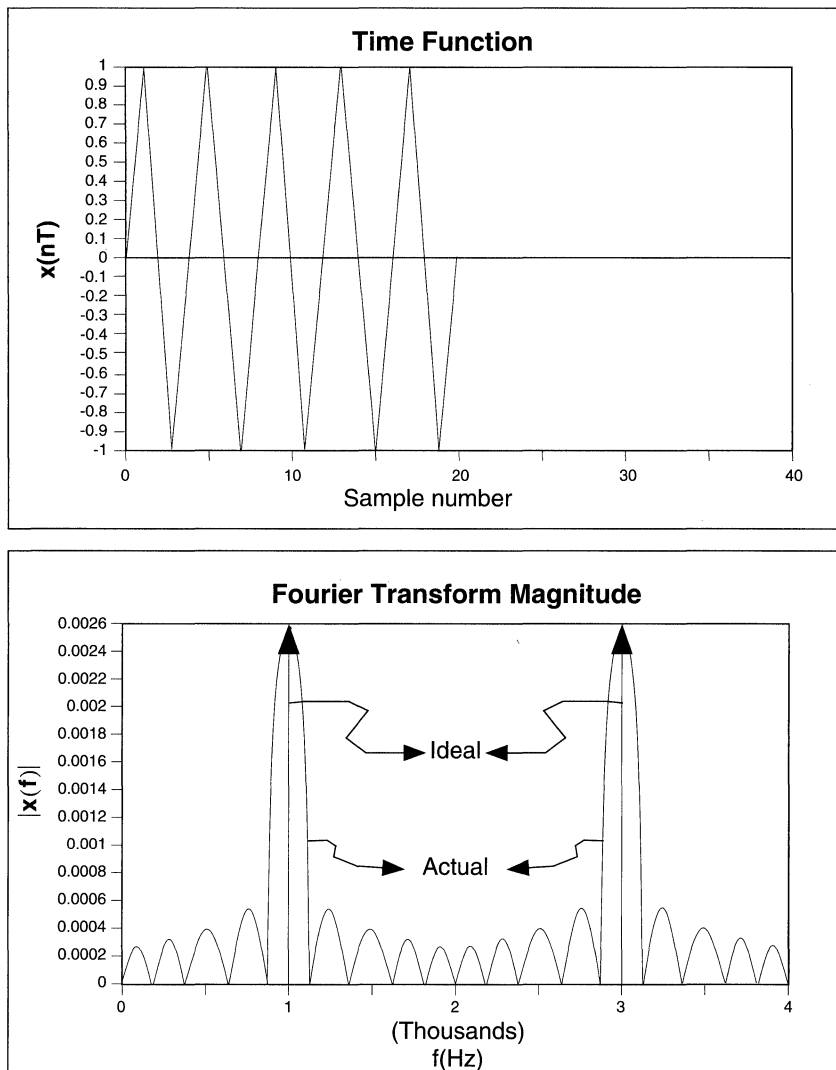


Fig. C.2 Windowing Effects When Windowing a Single Sine Wave

an appropriate windowing function can be chosen for a specific application. For an excellent summary of existing windowing functions and their properties, see Reference 2.

C.2.3 Sampling the Frequency Function

The windowed DTFT is now ready for machine computation, with one exception: the independent frequency variable f is still a continuous variable, and needs to be captured in discrete intervals, or sampled. Since the DTFT is periodic in the frequency

domain with period f_s , only values of f from 0 to f_s (the sampling frequency) need to be computed. Although there are similar arguments concerning the distance between successive frequency samples as in the case of time-sampling, it turns out that when the WDTFT is sampled every f_s/N Hz, fast algorithms for computing the transform can be derived. Note that in this case, the number of samples in the window (N) and the number of samples in the frequency domain (N) are equal. The resulting transform is called the discrete-time Fourier series (DTFS):

$$\tilde{X}_N(k) = T \sum_{n=0}^{N-1} \chi(nT) e^{-j\frac{2\pi}{N}nk} \quad (C.12)$$

The inverse DTFS is given by:

$$\chi_N(k) = \frac{1}{NT} \sum_{k=0}^{N-1} \tilde{X}_N(k) e^{j\frac{2\pi}{N}nk} \quad (C.13)$$

Keep in mind that the values of the frequency samples of f_k are equal to $[f_s/N] k$.

Note that many textbooks simply define the Discrete Fourier transform (DFT) $X_N(k)$:

$$X_N(k) = \sum_{n=0}^{N-1} \chi(nT) e^{-j\frac{2\pi}{N}nk} \quad (C.14)$$

with inverse transform:

$$\chi_N(n) = \frac{1}{N} \sum_{k=0}^{N-1} X_N(k) e^{j\frac{2\pi}{N}nk} \quad (C.15)$$

Obviously, the DFT and DTFS differ only by a scaling factor of T , making the spectrum independent of the sampling period. Consequently, explicit T dependence can be dropped from Eqn. (C.15).

Although the sequence $x_N(n)$ corresponds to the original sampled and windowed sequence $\chi(nT)$ for sampling instants 0 through $N-1$, the complete sampled sequence $\chi(nT)$ for any n cannot necessarily be recovered from it. Indeed, $x_N(n)$ appears to be

periodic with period N due to the periodicity of $e^{j\frac{2\pi}{N}nk}$, whereas the original sampled signal was not assumed to be periodic.¹

1. The error introduced in the time domain by sampling a frequency function is termed "aliasing in time" which is analogous to the "aliasing in frequency" caused by sampling a time function. (See SECTION C.2.1 *The Discrete-Time Fourier Transform (DTFT)*). That is, if a frequency spectrum is not sampled densely or closely enough, the signal constructed in the time domain through the inverse "discrete-frequency Fourier transform" will show some distortion.

This must be kept in mind in convolution-based applications, where the forward as well as inverse transforms are used; the incoming signal stream needs to be segmented, and the computed signal segments need to be pieced together to construct the complete output stream. Most basic textbooks on digital signal processing discuss techniques for piecing together the output stream (see Reference 3).

C.3 THE FAST FOURIER TRANSFORM

“Since there are two independent variables (time and frequency) in the Fourier transform, dividing (or decimating) the DFT into smaller ones can be done in two ways.”

C.3.1 Motivation

Upon closer examination of Eqn. (C.14), it becomes clear that for every frequency point, $N-1$ complex summations and N complex multiplications need to be evaluated. Since there are N frequency points to be evaluated, this gives a total of $N(N-1)$ complex sums, and N^2 complex multiplications. Counting two real sums for every complex one, and four real multiplications plus two real summations for every complex multiplication, gives a total of $4N^2 - 2N$ real summations and $4N^2$ real multiplications.

The above numbers grow rapidly for increasing N . For $N=1024$ (1024-point DFT), 4,194,304 real multiplications are required. If this is computed on a DSP56001/DSP56002 with a 27-MHz clock, it takes 0.31 second just to execute that many real multiplications. Since the DFT computation needs to be completed by the time the next 1024 data points are collected for real-time performance, the sampling rate is limited to a maximum of 3.3 kHz. Obviously, faster solutions are needed.

C.3.2 Divide and Conquer

A faster algorithm for computing the DFT can easily be derived. The principle behind this is very simple. As illustrated in Figure C.3, a square of half the linear dimension of a larger square has one-fourth the surface area. This is because the surface area is proportional to the square of the linear dimensions of the square. Similarly, the number of multiplications needed to compute the DFT is proportional to the square of the DFT's length (N). Thus, if we could replace the DFT over N points by two

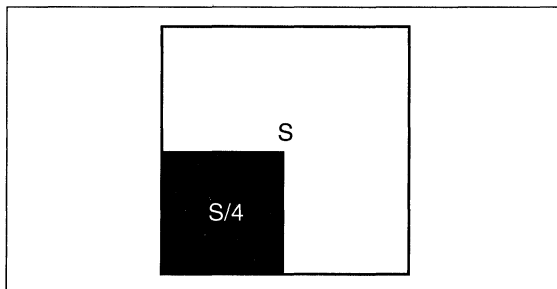


Fig. C.3 The FFT principle in layman's terms

DFTs over $N/2$ points, computations would be reduced in order of magnitude of 0.5 ($=0.25+0.25$).

Since there are two independent variables (time and frequency) in the Fourier transform, dividing (or decimating) the DFT into smaller ones can be done in two ways. We can attempt to represent an N -point transform in terms of DFTs over half the number ($N/2$) of time-samples. This approach is appropriately called the decimation-in-time or DIT approach. Alternatively, the N -point DFT can be represented in terms of DFTs with $N/2$ frequency samples. This approach is called the decimation-in-frequency or DIF approach.

C.3.3 The Decimation-in-Time and Decimation-in-Frequency Radix-2 Fast Fourier Transforms

It is easily shown that Eqn. (C.14) can be rewritten when N is even as:

$$X_N(k) = \sum_{r=0}^{(N/2)-1} x(2rT) e^{-j\frac{2\pi}{N}rk} + e^{-j\frac{2\pi}{N}(N/2)-1} \sum_{r=0}^{(N/2)-1} x[(2r+1)T] e^{-j\frac{2\pi}{N}rk} \quad (C.16)$$

As illustrated in Figure C.4, this expression shows how two $N/2$ -point DFTs can be combined to obtain one N -point DFT. If N is an integer power of 2, this process can be repeated, as shown in Figure C.5 and Figure C.6, until a simple, two-point DFT is obtained. This gives rise to the flow diagram of a DIT fast Fourier transform (FFT) as shown in Figure C.7, which represents a complete 8-point FFT computation.²

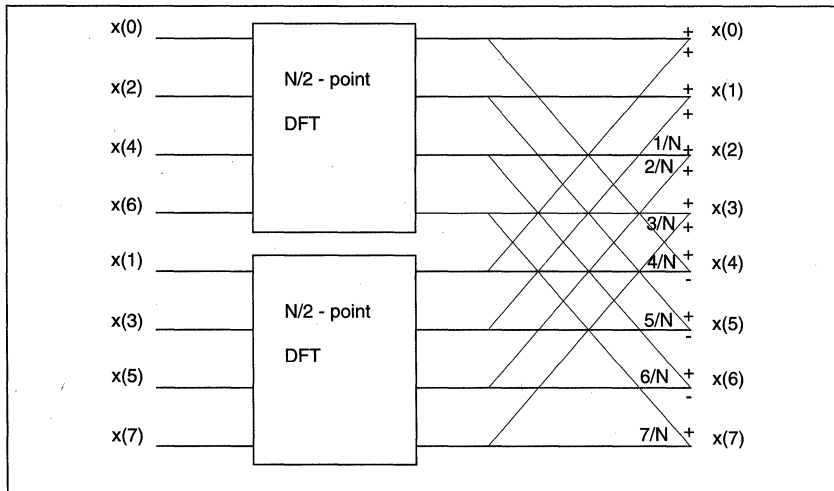


Fig. C.4 Decimation-in-Time of an N -Point FFT

2. k/N denotes multiplication by the "twiddle factors" $e^{-j\frac{2\pi}{N}k}$ throughout this document.

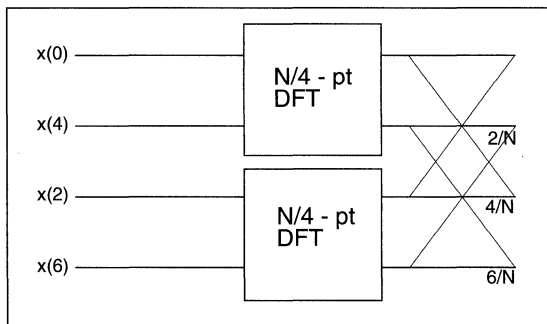


Fig. C.5 Decimation-in-Time FFT: Step Two

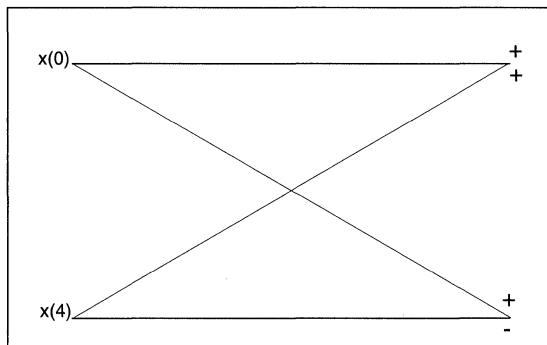


Fig. C.6 Decimation-in-Time FFT: Final Step (2-Point DFT)

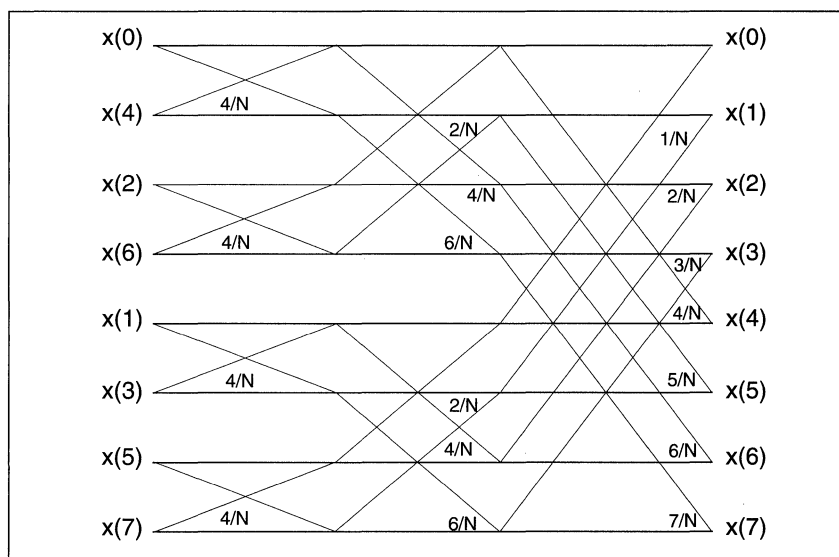


Fig. C.7 An 8-point, radix-2, Decimation-in-Time FFT

The basic flow diagram of Figure C.7 can be further simplified by rearranging the terms in the basic building block (the butterfly) as in Figure C.8. Also, it is seen from Figure C.7 that input samples no longer occur in normal, sequential order. When the indices are represented in their binary equivalent, however, the input samples appear in "bit-reversed" order. Figure C.10 shows how the diagram can be rearranged for normally-ordered inputs and bit-reversed outputs.

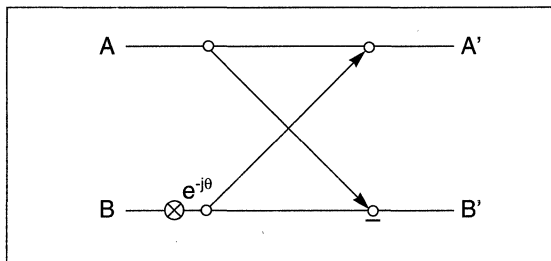


Fig. C.8 Rearrangement of the "butterfly" building block of the DIT FFT

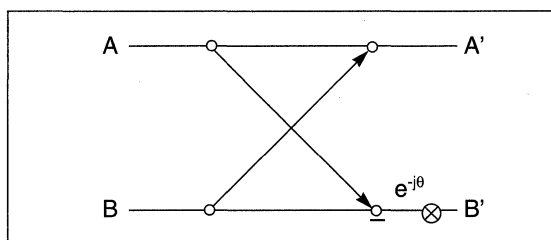


Fig. C.9 Rearrangement of the "butterfly" building block of the DIF FFT

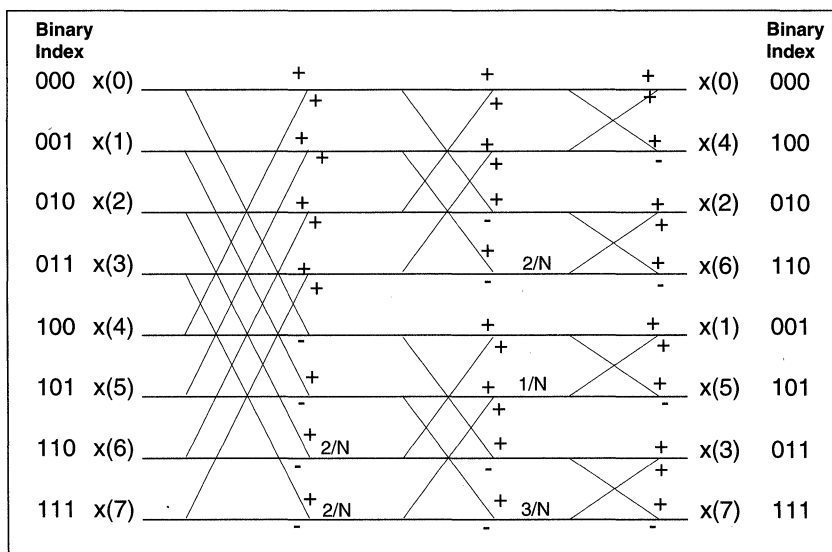


Fig. C.10 Rearrangement of the DIT computation of Figure C.8

Figure C.11 and Figure C.12 show how the DFT with N frequency points can be obtained in terms of DFTs with a smaller number of frequency samples (decimation-in-frequency FFT). Note that the basic building block (butterfly) is different than for the DIT case (see Figure C.12).

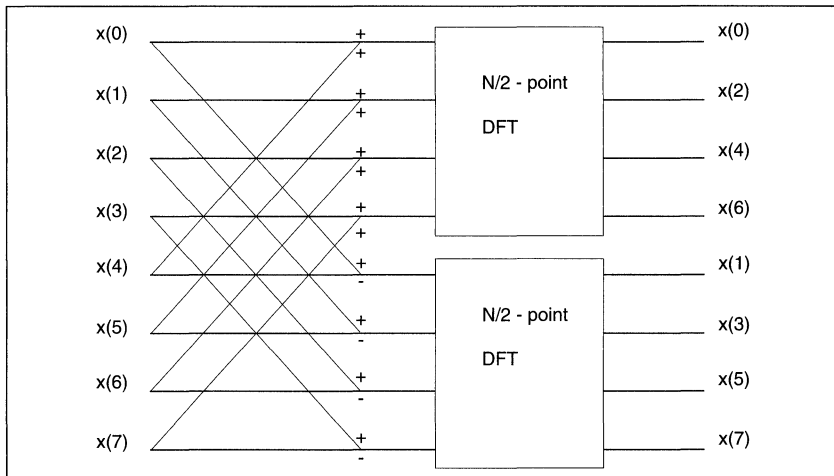


Fig. C.11 Decimation-in-Frequency concept

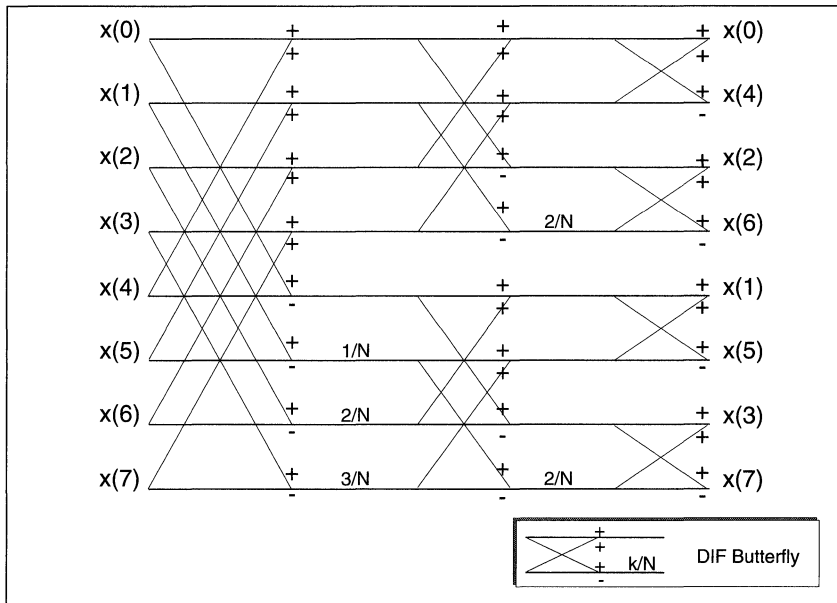


Fig. C.12 Complete 8-point radix-2 DIF FFT

C.3.4 The Decimation-in-Frequency Radix-2 Fast Fourier Transforms

If Eqn. (C.14) is decomposed from the frequency domain, we can show the following equations exist:

$$X_N(2k) = \sum_{r=0}^{(N/2)-1} [x(r) + x(r + N/2)] e^{-j \frac{2\pi r k}{N/2}} \quad (C.17)$$

$$= X_N(2k+1) = \sum_{r=0}^{(N/2)-1} [x(r) - x(r + N/2)] e^{-j \frac{2\pi r k}{N/2}} e^{-j \frac{2\pi r}{N}} \quad (C.18)$$

The decimation in frequency butterfly is shown in Figure C.11.

C.4 COMPLEX FFT ON THE MOTOROLA DSP FAMILY

"In general, doubling the points in butterflies of FFT reduces the number of groups in each pass and the number of passes."

C.4.1 Required Hardware Support for FFT Calculation

The basic building block of the DIT FFT routine is the butterfly computation shown in Figure C.8. Consequently, the architecture and instruction set of a DSP device should allow efficient computation of this basic butterfly. Since the butterfly consists of additions and multiplications, a hardware adder/subtractor and multiplier is crucial. The DSP56001/2 and the DSP56156 provide a multiplication and addition instruction, or MAC, which is beneficial to most DSP applications including FFT, with no increase in silicon cost. The DSP96002 supports FFT calculation capability by adding subtraction to the MAC function, which provides the multiplication, addition, and subtraction instruction, FMPY | ADD | SUB.

Since the butterfly calculation requires complex data, the architecture must easily support complex arithmetic. The input and output data to the butterflies are moved between the processor's arithmetic unit and memory. Consequently, efficient moves are needed.

DSP56001/2 and DSP96002 hardware features two data memory modules, X and Y. The real component and imaginary component of a complex number can be stored in the X and Y memory modules respectively. Also, the DSP56001/2 and the DSP96002 can perform dual reads and dual writes in one instruction cycle. In contrast, the DSP56156 has only one data memory module, X, where both real and imaginary components of the complex data are stored. To support complex number fetch, the DSP56156 provides dual memory read, where in one instruction, it reads the X memory twice if the specified address registers are used.

The overall FFT algorithm is an array of many such butterflies, and the size of the array depends upon the number of points (N) in the FFT. In order to write general

FFT routines (for any N of the power of 2), efficient implementation of the repetitive execution of the basic butterfly element is important. Although FFTs may be calculated on general-purpose microprocessors, typically, a great deal of software overhead is involved. A hardware solution, using hardware designed to efficiently implement the calculation of FFTs, would be generally preferred in a real-time system. The DSP56001/2, DSP96002, and DSP56156 feature a zero-overhead DO loop instruction. After the loop is set up (three instruction cycle time), each iteration takes no additional cost in overhead.

In real-life applications, time as well as frequency data is used in normal order, even though the diagram of Figure C.9 delivers the frequency data in bit-reversed order. Thus, an efficient method for bit-reversed addressing is needed while avoiding time-consuming software solutions that modify the addressing order. The DSP56001/2, DSP96002, and DSP56156 all feature a bit-reversed addressing mode.

Some FFT algorithms (for example, radix-4 FFT) require several registers to hold immediate results. The number of registers available on the DSPs is critical for computation intensive applications since storing and restoring intermediate results to and from memory will take more processing time than if the results are available in on-chip registers.

The input data (time samples) of the FFT is usually obtained from an external source such as an A/D converter. This data collection must occur in parallel with the FFT computation to make real-time performance possible. Consequently, a DSP device must provide easy interface with a variety of A/D converters, and must support low-overhead interrupt schemes which can load data from an external device with minimal impact on the FFT computation. The DSP56001/2, DSP96002, and DSP56156 all feature a variety of peripherals on chip. More details about real-time data acquisition are discussed in **SECTION C.7**.

The key points to implementing efficient FFT calculation using programmable DSPs are summarized below.

FFT calculation requires:

1. MAC or, ideally, FMPY || ADD || SUB instruction
2. Dual memory read and write in one instruction cycle
3. Zero-overhead loop instruction
4. Bit-reversed addressing mode
5. Sufficient number of registers
6. Fast I/O to provide real time data (in real-time applications)

C.4.2 Radix-2 DIT and DIF Butterflies

Theoretically, radix-2 decimation in time (DIT) butterflies and decimation in frequency (DIF) butterflies have the same computational complexity: three additions, three subtractions, and four multiplications. Since most DSPs have only one hardware multiplier, the minimum cycle time for multiplication for one DIT or DIF butterfly is four instruction cycles. However, on the DSP56001, a MAC instruction can implement

one multiplication and one addition in parallel in a single instruction cycle. Four of the six additions or subtractions in a DIT butterfly can be executed in parallel with four multiplications, and two more additions are required to finish the DIT butterfly calculation. Due to data dependence, a DIF butterfly can implement only two additions in parallel with two multiplications. Thus, one DIF butterfly calculation requires four multiplications plus four additions (see Figure C.15).

The DSP96002 features a special instruction, FMAY || ADD || SUB, which can implement either a DIT or a DIF butterfly in four instruction cycles. Although the DSP56156 has a MAC instruction, the lack of a dual memory write operation plus constraints on address pointer updates in dual memory read operations, causes the DIT butterfly and the DIF butterfly to both take eight instruction cycles.

In short, the Motorola DSP architecture implements the more efficient DIT butterfly, since it generates shorter cycle time than the DIF. The following discussions assume a radix-2 DIT, extending to radix-4 DIT in later sections.

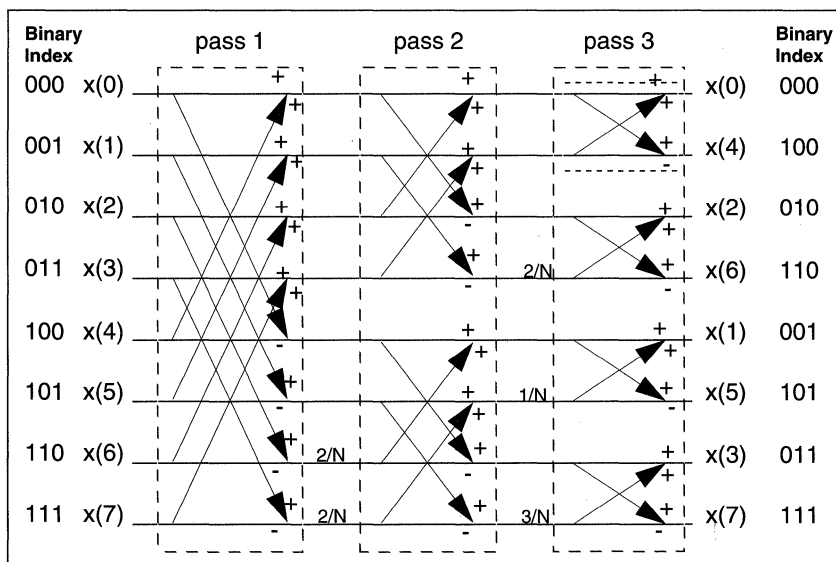


Fig. C.13 Grouping of Butterflies in the FFT Calculation

C.4.3 Complexity of a Radix-2 DIT FFT

The number of instructions required in a radix-2 DIT FFT is determined by the number of instructions in the butterfly core and the structural overhead of the DSP. If only arithmetic operations are counted in term of the multiplications and additions, a triple-nested implementation of the FFT (see next sections) requires the following number of instruction cycles for $N = 2^m$:

$$m \times N/2 \times \text{BFLY} \quad (\text{C.19})$$

where BFLY is number of instructions for calculating a complex input butterfly. For the DSP56001/2, the DSP96002, and the DSP56156, BFLY is 6, 4, and 8 respectively. On the DSP96002, for example, a 1024-point complex FFT needs $10 \times 512 \times 4 = 20,480$ instruction cycles.

C.4.4 Implementation on Motorola's DSP56001

C.4.4.1 DSP56001 Architecture The DSP56001 (see Reference 4) was the first member of the Motorola Digital Signal Processor line. It features 16.5 million instructions per second (MIPS) with a 33 MHz clock.

The data paths are 24 bits wide, thereby providing 144 dB of dynamic range. More importantly, intermediate results are held by a 56-bit accumulator which gives more accuracy in noise sensitive applications. The data ALU, address arithmetic units, and program controller operate in parallel so that an instruction pre-fetch, a 24x24-bit multiplication, a 56-bit addition, two data moves, and two address pointer updates using one of three types of arithmetic (linear, modulo, or bit-reversed) can be executed in one instruction. Three on-chip peripherals (Serial Communication Interface, Synchronous Serial Interface and Host Interface), a clock generator, and seven buses (three address, four data) make the overall system functionally complete and powerful. The architecture of DSP56001 is shown in Figure C.14.

C.4.4.2 DIT Butterfly Kernel on DSP56001 The parallel architecture and the instruction set of Motorola's DSP56001/2 lend themselves particularly well to the radix-2 DIT FFT computation. The DIT butterfly equations are programmed on Motorola's DSP56001/2 as given below:

$$\begin{aligned}
 A'_r &= A_r + B_r W_r + B_i W_i \\
 A'_i &= A_i + B_i W_r - B_r W_i \\
 B'_r &= 2A_r - A'_r \\
 B'_i &= 2A_i - A'_i
 \end{aligned} \tag{C.20}$$

where: **i** represents an imaginary component

r represents a real component

' symbolizes output items

The basic butterfly "core" is implemented by assembly language in Figure C.16. Note that the previous DSP56001/2 equations are written in this particular form such that the instruction to shift left and subtract accumulators (SUBL) can be used. This SUBL instruction allows efficient implementation of the DIT butterfly in a two-accumulator ALU.

The kernel shown in Figure C.16 executes in six instruction cycles, or a total of 12 clock cycles. This is made possible because of the parallel architecture of the DSP56001/2, which allows up to two data ALU operations (multiply/accumulate) in parallel with two data moves to/from memory and two pointer updates in a single instruction cycle. The dual data spaces X and Y with the appropriate X and Y buses

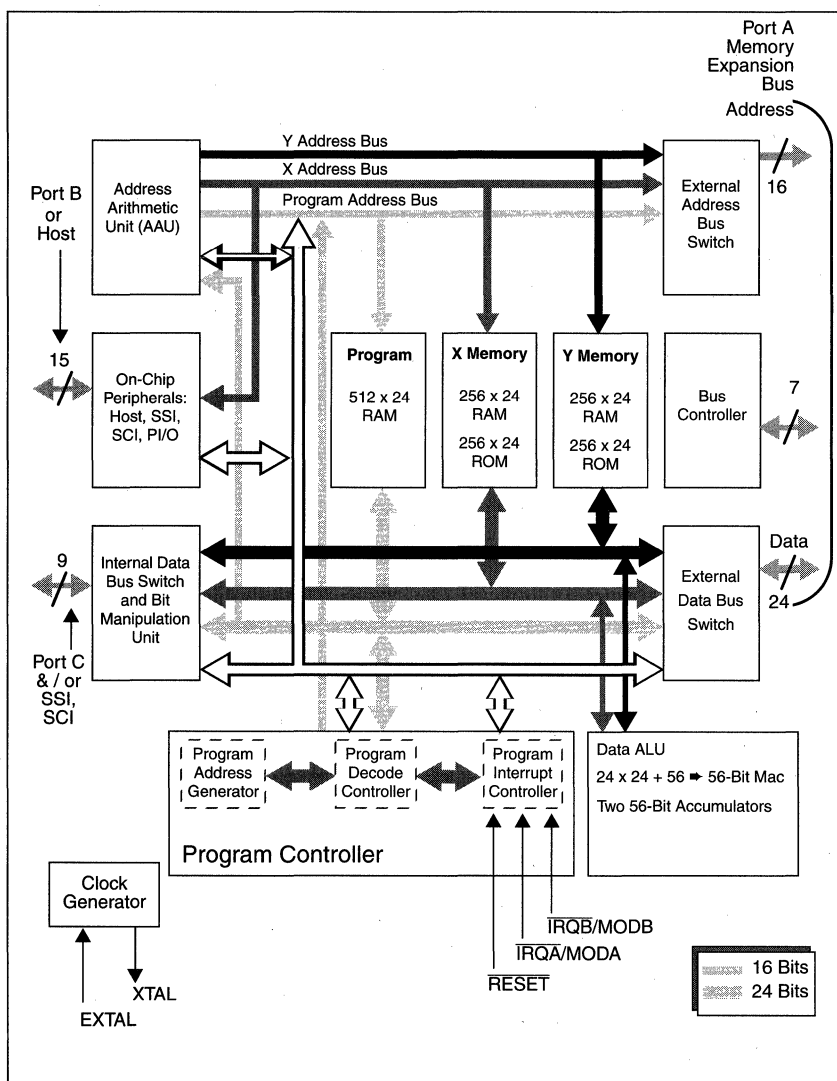


Fig. C.14 DSP56001 Architecture Block Diagram

are ideally suited for complex arithmetic; the real components are stored in X memory and the imaginary components are stored in Y memory.

The simplest way of combining all of the butterflies into a complete program is shown in Figure C.13. The FFT diagram is first divided into FFT passes. On each pass, the data is fetched from memory, the butterfly calculations are done, and the results are moved back out to memory. It is easily shown that there are $\log_2 N$ passes. Within each pass, the butterflies cluster in groups. From one pass to the next, the

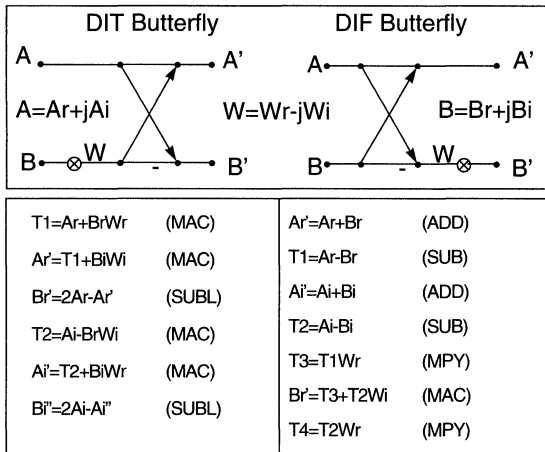


Fig. C.15 A radix-2 DIT butterfly needing less instruction cycles than a radix-2 DIF butterfly

```

;r0  ⤵  A
;r1  ⤵  B
;r4  ⤵  C
;r5  ⤵  D

mac    x1,y0,b    y:(r1)+,y1    ;Ai - BrWi ⤵ b,Bi ⤵ y1
macr   -x0,y1,b    a,x:(r5)+    y:(r0),a    ;Ai - BiWi + BiWr ⤵ b,Ai ⤵ a
subl   b,a        x:(r0),b    b,y:(r4)    ;2Ai - b ⤵ a,Ar ⤵ b
mac    -x1,x0,b    x:(r0)+,a    a,y:(r5)    ;Ar + BrWr ⤵ b,Ar ⤵ a
macr   -y1,y0,b    x:(r1),x1    ;Ar + BrWr + BiWi ⤵ b,Br ⤵ x1
subl   b,a        b,x:(r4)+    y:(r0),b    ;2Ar - b ⤵ a,Ai ⤵ b

```

Fig. C.16 The radix-2, DIT butterfly kernel on the DSP56001/2

number of groups doubles, while the number of butterflies per group is divided by two. Note that the twiddle factors are the same for all butterflies within each group, and that the order of the twiddle factors from one group to the next is bit-reversed. This is easily implemented on the DSP56001/2 by setting the appropriate modifier register (m6) equal to zero and the offset register (n6) equal to $N/4$ (= coefficient table size/2), such that the twiddle factors are addressed in bit-reversed manner.

This gives rise to the simple, triple-nested DO loop program shown in Figure C.17. The outer DO loop steps through passes, the middle loop goes through all of the groups within a pass, and the inner loop cycles through all of the butterflies inside a group. The DSP56001/2 is particularly well suited for looped program execution because it has hardware DO-loop capability. Once a loop is entered through the DO instruction, this loop is executed without any time penalty. The resulting program takes 40 words in program memory. This is the most compact implementation of the radix-2 DIT FFT. A 1024-point complex FFT using this code executes in 4.72 ms when using a 27-MHz clock.

Fig. C.17 A Simple, Triple-Nested DO Loop Radix-2 DIT FFT on DSP56001/2

```

;This program originally available on the Motorola DSP bulletin board.
;It is provided under a DISCLAIMER OF WARRANTY available from
;Motorola DSP Operation, 6501 Wm. Cannon Drive W., Austin, Tx., 78735.
;
;Radix 2, In-Place, Decimation-In-Time FFT (smallest code size).
;
;Last Update 30 Sept. 86 Version 1.1
;
fftr2a    macro    points, data, coef
fftr2a    ident    1,1
;
;Radix 2 Decimation in Time In-Place Fast Fourier Transform Routine
; Complex input and output data
;   Real data in X memory
;   Imaginary data in Y memory
; Normally ordered input data
; Bit reversed output data
; Coefficient lookup table
;   -Cosine values in X memory
;   -Sine values in Y memory
;
;Macro Call - fftr2a points,data,coef
;
; points      number of points (2-32768, power of 2)
; data        start of data buffer
; coef        start of sine/cosine table
;
;Alters Data ALU Registers
; x1 x0 y1 y0
; a2 a1 a0 a
; b2 b1 b0 b
;
;Alters Address Registers
; r0 n0 m0
; r1 n1 m1
;      n2
;
; r4 n4 m4
; r5 n6 m5
; r6 n6 m6
;
;Alters Program Control Registers
; pc sr
;Uses 6 locations or System Stack
;Latest Revision   September 30, 1986
;r0 points to A
;r1 points to B
;r4 points to C
;r5 points to D
;r6 points to twiddle factor
;   move # points/2,n0      ;initialize butterflies per group
;       move # 1,n2         ;initialize groups per pass
;       move # points/4,n6  ;initialize C pointer offset
;       move #-1,m0         ;initialize A and B address modifiers
;       move m0,m1          ;for linear addressing
;       move m0,m4
;       move m0,m5
;       move #0,m6          ;initialize C address modifier for
;                           ;reverse carry (bit-reversed) addressing
;
;Perform all FFT passes with triple nested DO loop
;
;   d0      #(axcvi(axlog(points)/(axlog(2)+0.5)_end_pass
;   move    #data,r0        ;initialize A input pointer
;   move    r0,r4           ;initialize A output pointer
;   lua     (r0)+n0,r1[      ;initialize B input pointer
;   move    #coef,r6        ;initialize C input pointer
;   lua     (r1)-,r5         ;initialize B output pointer

```

```

move    n0,n1                                ;initialize pointer offsets
move    n0,n4
move    n0,n5
do      n2,_end_grp
move    x:(r1),xly:(r6),y0                    ;lookup -sine and
                                              ;-cosine values
move    x:(r5),a      y:(r0),b                ;preload data
move    x:(r6)+n6,x0                          ;update C pointer

do      n0,_end_bfy
mac     x1,y0,b      y:(r1)+,y1                ;Radix 2 DIT
                                              ;butterfly kernel

macr    -x0,y1,b      a,x:(r5)+      y:(r0),a
subl    b,a          x:(r0),b      b,y:(r4)
mac     -x1,x0,b      x:(r0)+,a      a,y:(r5)
macr    -y1,y0,b      tx:(r1),x1
subl    b,a          b,x:(r4)+      y:(r0),b
_end_bfy
move    a,x:(r5)+n5 y:(r1)+n1,y1              ;update A and B pointers
move    x:(r0)+0,x1 y:(r4)+4,y1

_end_grp
move    n0,b1
lsr     b          n2,a1                      ;divide butterflies per group by two
lsr     a          b1,n0                      ;multiply groups per pass by two
move    a1,n2
_end_pass
endm

```

C.4.5 Implementation on Motorola's DSP96002

C.4.5.1 DSP96002 Architecture DSP96002 is a 32-bit floating-point digital signal processor with 20 million instructions execution per second using a 40 MHz clock. The data ALU provides full conformance with the IEEE 754-1985 Standard for Single Precision Binary Floating-Point Arithmetic. Single Extended precision with a 32-bit mantissa and 11-bit exponent is also implemented. The data ALU, AGU, and program controller operate in parallel within the CPU so that an instruction pre-fetch, up to three floating point operations, two data moves, and four address pointer updates using one of three types of arithmetic (linear, modulo, and reverse carry) can all be executed in one instruction cycle.

Also, an on-chip dual channel DMA controller generates two addresses, using one of the three types of address update arithmetic so that a memory-to-memory or memory-to-peripheral transfer can occur in parallel with the CPU operation during each instruction cycle. Host interface circuitry on each port provides a flexible slave interface to external processors and/or DMA controllers for easy design of a multi-master system. Designed primarily for image processing, real-time data acquisition, sonar signal processing, radar signal processing, medical image analysis, and video compression, the DSP96002 has the widest data bandwidth of any DSP currently on the market. A special FMAY | | ADD | | SUB instruction makes FFT calculations extremely fast on the DSP96002.

C.4.5.2 DIT Butterfly Kernel on DSP96002 The butterfly equations implemented in the radix-2, DIT FFT on DSP96002 are the following:

$$\begin{aligned}
 A'_r &= A_r + B_r W_r + B_i W_i \\
 A'_i &= A_i + B_i W_r - B_r W_i \\
 B'_r &= A_r - (B_r W_r + B_i W_i) \\
 B'_i &= A_i - (B_i W_r - B_r W_i)
 \end{aligned}
 \tag{C.21}$$

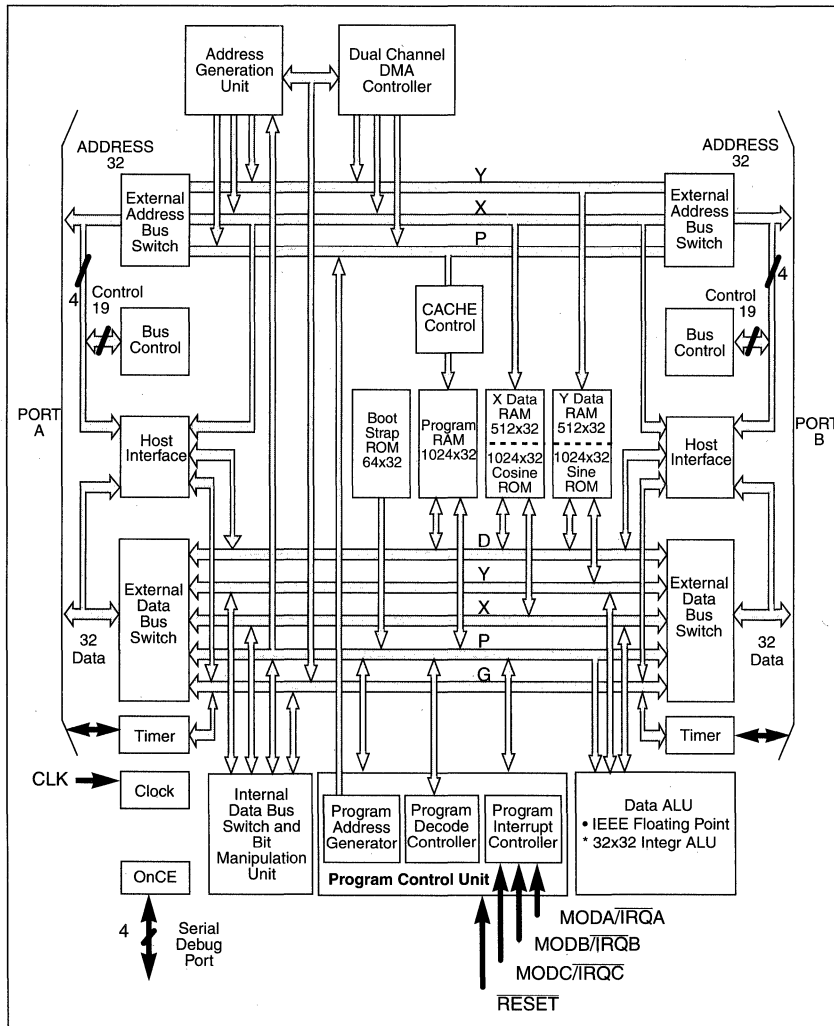


Fig. C.18 DSP96002 Architectural Block Diagram. Two symmetric bus expansion ports with two channel DMA controller that blow away the speed limit on external memory access and data I/O.

where: j represents an imaginary component

r represents a real component

' symbolizes output items

The implementation of this basic butterfly in DSP96002 assembly language code is shown in Figure C.19. The kernel in Eqn. (C.21) executes in four instruction cycles, or eight clock cycles. Since four real multiplications are needed, and only one real multiplier is available, this is the most efficient implementation possible. In addition to the features available on the DSP56001/2, this efficient execution is obtained by the FADDSUB instruction which delivers the sum and the difference of two operands, in parallel with a multiplication and two data moves. With this feature, a total of three floating-point operations can be executed in one instruction cycle, resulting in a peak performance of 60 million floating-point operations per second (MFLOPS) with a 40-MHz clock.

The triple-nested DO loop routine, which computes the radix-2, DIT FFT on the DSP96002 takes only 30 words in program memory. A 1024-point complex FFT is executed in only 2.31 ms, assuming a 27-MHz clock.

Fig. C.19 The Radix-2, DIT FFT Butterfly Kernel on the DSP96002

```

;r0 ➔ A
;r1 ➔ B
;r4 ➔ C
;r5 ➔ D

fmpy d8,d6,d fadd.s d3,d0 x:(r0),d4.s d2.s,y:(r5)+ ;Br*sin ➔ d2
                                                    ;Bj*sin + Br*cos ➔ d0
                                                    ;Ar ➔ d4,Dj ➔ mem.

fmpy d8,d7,d3 faddsub.sd4,d0 x:(r1)+,d6.s d5.s,y:(r4)+ ;Bj*sin ➔ d3
                                                    ;Ar + Br1 ➔ d0
                                                    ;Ar - Br1 ➔ d4
                                                    ;Br ➔ d6
                                                    ;Cj ➔ mem.

fmpy d9,d6,d0 fsub.sd1,d2 d0.s,x:(r4) y:(r0) + d5.s ;Br*cos ➔ d0
                                                    ;Br*sin - Bj*cos ➔ d2
                                                    ;Cr ➔ mem.
                                                    ;Aj ➔ d5

fmpy d9,d7,d1 faddsub.sd5,d2 d4.s,x:(r5) y:(r1),d7.s ;Bj*cos ➔ d1
                                                    ;Aj + Bj1 ➔ d2
                                                    ;Aj - Bj1 ➔ d5
                                                    ;Dr ➔ mem.
                                                    ;Bj ➔ d7

```

C.4.6 Implementation on Motorola's DSP56156

C.4.6.1 DSP56156 Architecture The DSP56156 is the most recent addition to the Motorola DSP line. This 16-bit fixed-point number DSP is designed primarily for speech coding and telecommunication. The on-chip sigma-delta codec functions as a

bridge between the analog and digital world. The on-chip phase-locked-loop (PLL) reduces clock noise to a minimum. Operating at 60 MHz, the DSP56156 can execute 30 million instructions per second with two kilowords (2k) on-chip data RAM (which is

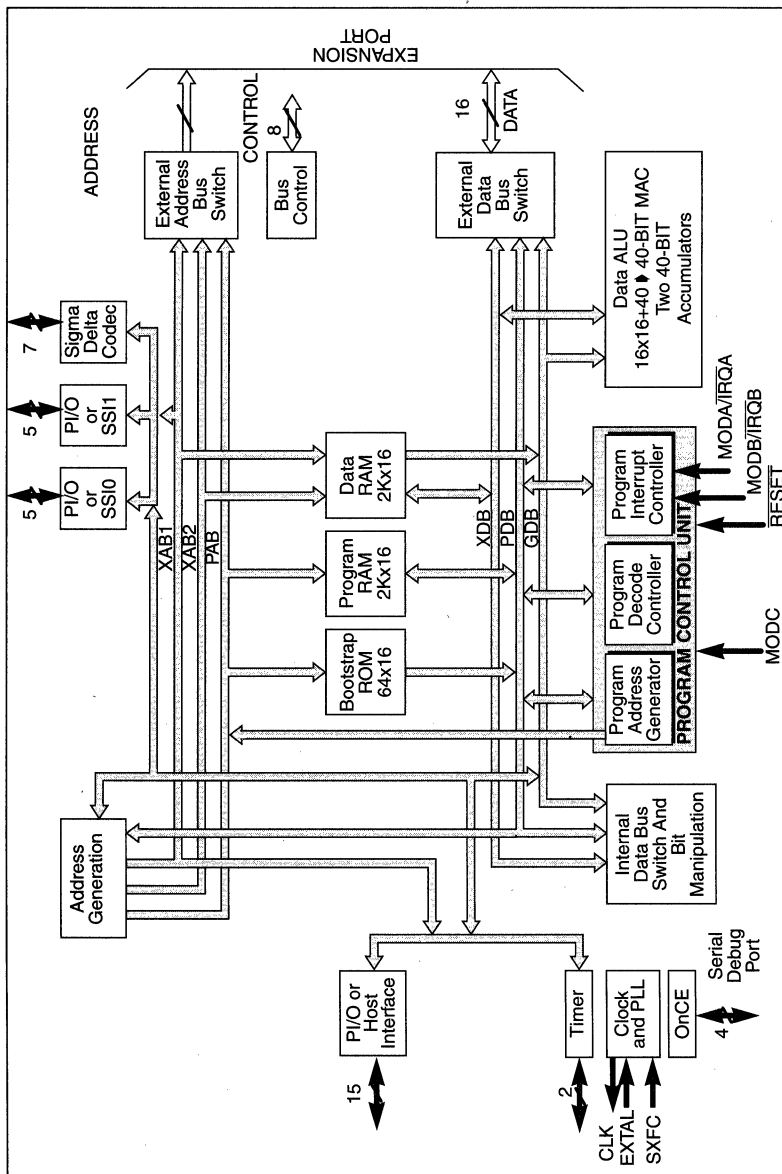


Fig. C.20 DSP56156 architectural block diagram. Only four address registers are available. On the single data memory space, only a dual-read per one instruction is allowed.

four times larger than DSP56001's) and four address registers. Since the DSP56156 is designed for the digital cellular phone, its limited instruction operation codes must focus on telecommunication capability, and some of its advanced addressing modes and instructions that accelerate FFT calculation must be compromised due to the smaller instruction words.

Although only one memory module can be accessed in a single instruction cycle, the DSP56156 *does* support dual memory reads. However, it does *not* support dual memory writes in a single instruction cycle. Four address registers and a single write per instruction may slow down FFT performance on DSP56156, but having 2k on-chip data memory may compensate for a portion of the performance loss, i.e. dual on-chip memory reads may save time equivalent to four instruction cycles if the number of data points is between 256 and 1024 points.

C.4.6.2 DIT Butterfly Kernel on DSP56156 The butterfly equation for the DSP56156 is the same as the DIT butterfly equation for the DSP56001/2 as shown in Eqn. (C.20). However, two more instructions are required in the DSP56156 butterfly than the DSP56001/2 because of its lack of a dual-write operation and its constraints on the address register mode. Figure C.21 shows the DSP56156 assembly language code of the butterfly core.

Fig. C.21 The butterfly core of the DSP56156. Notice that a single write operation paralleling with an instruction always occupies a whole data move field.

mpy	x0,y0,b	a,x:(r2)+		;b=W _r Br, save prev. Bi', r2 -> Br
macr	x1,y1,b	x:(r0)+n0,a		;b=W _r Br+WiBi, a=Ar
add	a,b			;b=Ar+W _r Br+WiBi=Ar'
subl	b,a	b,x:(r0)+		;a=2Ar-Ar'=Br', save Ar', r0 pt to Ai
mpy	-y1,x0,b	a,x:(r2)+		;b=-WiBr, save Br', r2 pt to Bi
macr	y0,x1,b	x:(r0)+n0,a	x:(r3)+,x0	;b=-WiBr+W _r Bi, a=Ai, x0=next Br
add	a,b	x:(r3)+,x1		;b=Ai-WiBr+W _r Bi=Ai', x1=next Bi
subl	b,a	b,x:(r0)+		;a=2Ai-Ai'=Bi', save Ai', r0-> next Ar

C.4.7 Scaling for Fixed-Point Processors (DSP56001/2 and DSP56156)

Whenever mathematical algorithms are implemented in digital hardware, note that results are obtained with finite precision. The precision is generally limited by the number of bits used in the number representation, and depends on how the arithmetic limits its results to those bits. The user must use care to prevent overflows in the FFT outputs of fixed-point DSPs. Scaling via shifting or dividing can keep input data or intermediate results within the correct range, while maintaining maximum precision on the outputs.

C.4.7.1 Scaling at the Input – Guard Bits Since data length grows with each pass, overflow can occur at any pass if there is no scaling in the input of a fixed point number DSP. The magnitude of the output by the DIT butterfly defined in Eqn. (C.20) will grow an average of one bit on the output in each pass. This is based on the observation that output A' (a complex output) can be rewritten as $A' = A + B \times W$ where A', A, B, and W are complex numbers. Since $W = e^{j\theta}$, it has a unit magnitude.

The complex operation $B \times W$ simply rotates B according to θ and causes no magnitude growth. Complex addition is the only chance in a single butterfly calculation to make the output magnitude grow larger than a value of one. One addition can cause growth of one bit. Therefore, for $N = 2^m$ points of the FFT, m passes are required, i.e., m times a potential worst case magnitude doubling. However, the twiddle factor will reach its maximum magnitude when $\theta = \pi/4$. For this case, the maximum magnitude growth is 2.4 bits on real and imaginary components. Fortunately, only two groups of butterflies in each pass will use the maximum twiddle factors. No butterflies use the maximum twiddle factors twice within an entire FFT calculation. This mutually exclusive characteristic is the base upon which block floating point arithmetic is designed.

To prevent overflows in the FFT calculations, the input data should keep m zeros in the significant part so that growth bits will not get lost during the overflow. The m zeros are called "guard bits." To obtain sufficient guard bits, divide the input data words by N . For example, if the DSP56001 is implementing a 1024-point complex FFT, 10 guard bits are inserted into the most significant bits of the 24-bit data word, resulting in 14 bits of actual information. But on the 16-bit DSP56156, only 6 bits contain actual information after 10 guard bits are inserted. This may make the signal-to-noise ratio unacceptably low. This method of scaling the input data is simple and effective on a smaller FFT or on a large data word processor like the DSP56001. For a larger FFT or a small data word processor, an alternative method discussed in the next subsection may result in improved signal-to-noise ratio with some trade-offs.

C.4.7.2 Scaling During the Passes – Auto-Scaling and Block Floating-Point

Scaling in the input truncates valuable information contained in data words by shifting input data right by m -bits. $6.02 \times m$ dB have already been lost before the start of the FFT calculations. As indicated in the last subsection, an average of one bit word growth occurs in each pass. Another way to prevent overflow in the FFT calculation is to scale down the output of the butterfly by two at each pass, regardless of whether or not an overflow occurs. Since the scaling down at the output is automatically carried out to the next pass, the amount of scaling down is known beforehand. To obtain the true FFT output, simply multiply each output by N . This method is simple and has better signal-to-noise ratio than the scaling in the input method. But some passes may not have bit growth or overflows, so excessive scaling may occur, and automatic scaling may cause some information to be lost.

A more aggressive method treats one pass as one block of data, and assigns an exponent for each block. If bit growth occurs, the method scales down the output by one bit and increases the exponent by one. At the end of the FFT, the same number of scaling up operations must be carried out. In the DSP56156/DSP56002, the scaling bit (bit 7 in the status register) eases implementation of this method. The scaling bit is referred to as a "sticky" bit because once set, it retains its status until the next read of the status register. Five more instructions are added to the end of each pass to check the scaling bit in the DSP56002 and DSP56156, and to update the exponent of the complex FFT. (See program FFTBF.asm on the Motorola DSP bulletin board; Dr. BuB.) Among the methods discussed here, the sticky bit method gives the best signal-to-noise ratio.

C.4.8 Twiddle Factors and On-Chip ROM

C.4.8.1 Twiddle Factors for Decimation-in-Time Twiddle factors, $W_N^k = e^{-j2\pi k/N}$, are coefficients used in FFT calculations. For normal order input radix-2 decimation-in-time FFT, the twiddle factors are always fetched in bit-reversed order, i.e.

$$W_N^0, W_N^{(N/2)-1}, W_N^{N/4}, W_N^{N/8}, W_N^{(3N)/8}, \dots, W_N^{(N/4)-1}$$

Note that for an N point radix-2 FFT, two input data words share one twiddle factor, and the bit-reversed order of the twiddle factor is based on N/2 points.

C.4.8.2 Sine Table on the DSP56001/2 When the data-ROM-enable (DE) bit in the OMR register of the DSP56001/2 is set, the Y memory from \$100 to \$1FF contains a 256-point full cycle sine-wave, and each data entry has 24-bit accuracy. As mentioned in the last subsection, for an N point FFT, N/2 complex coefficient twiddle factors are required, and these N/2 twiddle factors are a half cycle of the sine and cosine waveforms. Since only a 256-point full cycle sine-wave is stored in the DSP56001/2 data ROM, the maximum FFT length utilizing only internal twiddle factors is one full cycle of the sine table, 256 points. However, a FFT larger than 256 points can still be implemented utilizing the on-chip sine table by calling this internal ROM during the first several passes and the first several groups in the last pass. Because DIT and normal input order FFT require bit-reversed sine and cosine tables, the DSP must be in the bit-reversed addressing mode when the on-chip sine table is invoked. A common set-up for addressing this table is:

$$\begin{aligned} r6 &= \$100 \\ n6 &= \$40 \\ m6 &= 0 \end{aligned}$$

To address the cosine table in the FFT calculation, the following relation between sine and cosine is utilized:

$$\cos(x) = \sin(x + \pi/2) \quad (\text{C.22})$$

Another address pointer, for example r2, is used to point to the correct location.

$$\begin{aligned} r2 &= \$140 \\ n2 &= \$40 \\ m0 &= 0 \end{aligned}$$

This set-up can be applied for all FFTs up to 256 points with length equaling a power of two, 2^N .

C.4.8.3 Sine and Cosine Tables on the DSP96002 The on-chip ROM of the DSP96002 features sine and cosine tables. When the DE bit is set to 1, X and Y memory from \$400 to \$7FF contain 512-point cosine and sine tables respectively. There-

fore, the maximum data length of the FFT without utilizing external twiddle factors is 512 points. The addressing set-up is similar to that of the DSP56001:

```

r6 = $400
n6 = $100
m6 = 0

```

Only one set of address registers is required on the DSP96002 to access both sine and cosine values.

C.4.9 Bit-Reversed Addressing

All Motorola DSPs feature a bit-reversed or inverse-carry addressing mode to accelerate FFT calculations. When bit-reversed addressing is enabled, an additional temporary data buffer is required to hold normal order outputs since bit-reversing on the fly is not an in-place method of FFT calculation. In some situations, the memory space used is more critical than the time used. To reduce the requirement for space in the second buffer, an in-place bit-reversed method is preferred. However, there is a time penalty for space-saving since the in-place bit-reversal must be carried out after the FFT is done. Program BITREVTWD56.asm on the Motorola DSP bulletin board (Dr. BuB) presents an example of in-place bit-reverse for DSP56001/2. The algorithm that performs conversion from bit-reversed order to normal order addressing is presented in Figure C.22.

Fig. C.22 In-place bit-reversed to normal order conversion

```

normal_order=output_pointer;
bitrev_order=data_buffer;
for (i=0;i<N;i++){
    normal_order++;
    bitrev_order+=N/2;
    \* suppose bit reverse address available *\
    if (normal_order< bitrev_order)
        data[normal_order]=data[bitrev_order]
}

```

C.4.10 Implementation of a Radix-4 DIT FFT on DSP96002

In general, doubling the points in butterflies of FFT reduces the number of groups in each pass and the number of passes. A radix-4 butterfly accepts four complex inputs, thus, the number of butterflies in a pass is $N/4$, and the number of passes is $\log_4(N)$. However, the number of instructions required in the radix-4 butterfly is three times that of the radix-2 butterfly. If the number of the instructions used in a radix-4 butterfly is four or more times than that of the radix-2's on a processor, there is really no advantage to adapting the radix-4 FFT on such a processor. Because the outputs or inputs of a radix-4 FFT might be digit-reversed order which is not being supported by any DSPs in the market. A software routine has to be used for converting digit-reversed order data to the normal one.

C.4.10.1 Radix-4 DIT Butterfly Core The butterfly equations for a radix-4 DIT FFT can be derived directly from two stages of radix-2 DIT butterflies, which are plotted in Figure C.23. There are four butterflies with four twiddle factors involved in the calculation. In the first pass, pass x , two butterflies are in the same group (the twiddle factors for a group are identical). In the second pass, pass $x+1$, two adjacent butterflies share one twiddle factor but differ by $-j$. (See **SECTION C.5.1 Optimization**.)

There are four complex multiplications required which can be reduced to three by combining them into a radix-4 butterfly. Eqn. (C.23) shows two-stage radix-2 butterfly calculations.

$$\begin{aligned} A' &= A + CW^c + (BW^b + DW^cW^b) \\ B' &= A + CW^c - (BW^b + DW^cW^b) \\ C' &= A - CW^c - j(BW^b - DW^cW^b) \\ D' &= A - CW^c + j(BW^b - DW^cW^b) \end{aligned} \quad (C.23)$$

Let $W^bW^c = W^d$, which gives us Eqn. (C.24). A new flow diagram for radix-4 DIT FFT results as shown in Figure C.24. Three twiddle factors are needed. W_a and W_b originally come from the radix-2 DIT FFT; W_c is new for the radix-4 FFT. Note that the radix-4 DIT butterfly accesses $1/3$ more twiddle factors than the radix-2 does.

$$\begin{aligned} A' &= A + CW^c + (BW^b + DW^d) \\ B' &= A + CW^c - (BW^b + DW^d) \\ C' &= A - CW^c - j(BW^b - DW^d) \\ D' &= A - CW^c + j(BW^b - DW^d) \end{aligned} \quad (C.24)$$

Since each butterfly takes four complex inputs and generates four complex outputs, the number of groups in a pass is reduced to $N/4$. Also, the number of passes is

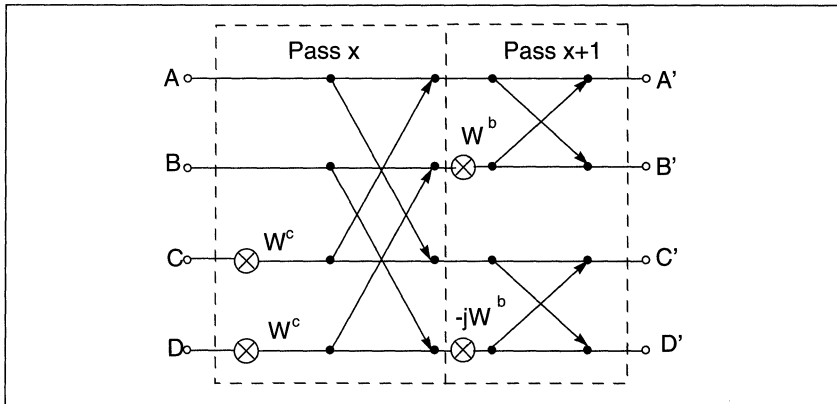


Fig. C.23 A flow diagram of two stages in a radix-2 DIT butterfly—four complex multiplications are involved in the computation.

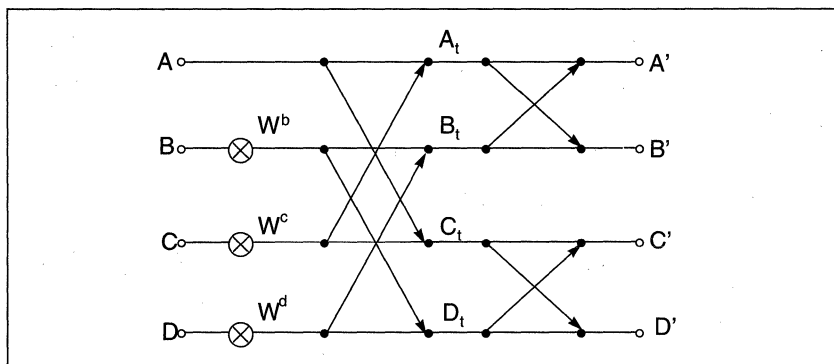


Fig. C.24 A flow diagram of a Radix-4 DIT butterfly; 12 multiplications and 22 additions or subtractions are required.

reduced to $\log_4(N)$. Theoretically, the lower boundary for radix-4 DIT FFT is:

$$\text{TRIV} \times N/4 + (\log_4(N) - 1) \times N/4 \times \text{BFLY}$$

Twelve multiplications, fourteen additions, and eight subtractions are required for a radix-4 DIT butterfly, as Eqn. (C.25) illustrates.

$$\begin{aligned}
 A_{tr} &= A_r + C_r W_r^c - C_i W_r^c \\
 A_{ti} &= A_i + C_r W_r^c + C_i W_r^c \\
 B_{tr} &= B_r W_r^b + D_r W_r^d - B_i W_r^b - D_i W_r^d \\
 B_{ti} &= B_r W_r^b + D_r W_r^d + B_i W_r^b + D_i W_r^d \\
 C_{tr} &= A_r - C_r W_r^c + C_i W_r^c \\
 C_{ti} &= A_r - C_i W_r^c - C_r W_r^c \\
 D_{tr} &= B_r W_r^b - D_r W_r^d - B_i W_r^b + D_i W_r^d \\
 D_{ti} &= B_r W_r^b - D_r W_r^d + B_i W_r^b - D_i W_r^d \\
 A_{r'} &= A_{tr} + B_{tr} \\
 A_{i'} &= A_{ti} + B_{ti} \\
 B_{r'} &= A_{tr} - B_{tr} \\
 B_{i'} &= A_{ti} - B_{ti} \\
 C_{r'} &= C_{tr} + D_{tr} \\
 C_{i'} &= C_{ti} - D_{tr} \\
 D_{r'} &= C_{tr} - D_{ti} \\
 D_{i'} &= C_{ti} + D_{tr}
 \end{aligned} \tag{C.25}$$

Fig. C.25 Radix-4 DIT Butterfly takes 17 instructions on the DSP96002

```

;r0->A, r4->B, r1->C, r6->D;
;r1->A', r3->B', r5->C', r7->D';
;n0=n4=4, n4=2;
;n2=n3=n5=n7=N/8.

```

		move		x: (r4)+n4, d3.s	y: ,d5.s
		move		x: (r4)+n4, d1.s	y: ,d2.s
		faddsub.s	d1, d3	x: (r0), d7.s	
		faddsub.s	d5, d2	x: (r1), d0.s	d1.s, y: (r7)
		faddsub.s	d7, d0	d3.s, d4.s	y: (r1)+n1, d1.s
		faddsub.s	d7, d5	x: (r4), d6.s	y: (r0)+n0, d3.s
		faddsub.s	d0, d4	d7.s, x: (r3)	y: (r4)+n4, d7.s
do	#N/4, _end_r4				
		faddsub.s	d3, d1	x: (r6)+, d9.s	y: ,d8.s
fmpy.s	d6, d9, d5			d5.s, x: (r7)	
fmpy	d7, d8, d3	faddsub.s	d1, d2	d4.s, x: (r5)	d3.s, d4.s
fmpy	d6, d8, d1	fadd.s	d5, d3	d0.s, x: (r2)+n2	d1.s, y:
fmpy.s	d7, d9, d5			x: (r6)+, d9.s	y: ,d8.s
		fsub.s	d1, d5	x: (r4)+n4, d6.s	y: ,d7.s
fmpy.s	d6, d9, d1				y: (r7), d0.s
fmpy	d7, d8, d2	faddsub.s	d4, d0	d2.s, y: (r5)+n5	
fmpy	d6, d8, d0	fadd.s	d2, d1	x: (r1), d6.s	d0.s, y: (r7)+n7
fmpy	d7, d9, d2	faddsub.s	d1, d3	x: (r6)+, d9.s	y: ,d8.s
fmpy	d6, d9, d0	fsub.s	d0, d2	y: (r1)+n1, d7.s	
fmpy	d7, d8, d3	faddsub.s	d5, d2	d3.s, d4.s	d4.s, y: (r3)+n3
fmpy	d7, d9, d1	fadd.s	d3, d0	x: (r0), d7.s	d1.s, y: (r7)
fmpy	d6, d8, d3	faddsub.s	d7, d0		
		faddsub.s	d7, d5		
		faddsub.s	d0, d4	d7.s, x: (r3)	y: (r4), d7.s
		fsub.s	d3, d1	x: (r4)+n4, d6.s	y: (r0)+n0, d3.s
_end_r4					
		faddsub.s	d3, d1	d5.s, x: (r7)	
		faddsub.s	d1, d2	y: (r7), d6.s	
		move		d0.s, x: (r2)	d1.s, y:
		faddsub.s	d3, d6	d4.s, x: (r5)	d2.s, y:
		move			d6.s, y: (r7)
		move			d3.s, y: (r3)

For example, if there are 1024-point complex inputs, $8 \times 256 + 4 \times 256 \times 14 = 16,384$ instructions may be required to improve performance by 11% if compared with 1024-point radix-2 DIT FFT. Here assume, $\text{TRIV} = 8$ and $\text{BFLY} = 14$ since eight $\text{ADD} \parallel \text{SUB}$ and six ADD instructions are theoretically required for such a butterfly calculation. One important fact is that BFLY (the number of instruction cycles for butterfly calculation) in a radix-4 DIT FFT must be less than 16, otherwise, there is no advantage for using radix-4 over radix-2. Due to an insufficient number of operations code, $\text{FMPY} // \text{ADD} // \text{SUB}$ instruction only works with destination registers D0 to D3 on the DSP96002.

C.4.10.2 Radix-4 DIF Butterfly Core Using the same derivation, a radix-4 DIF butterfly can be obtained. Although the number of multiplications and additions is the same as the radix-4 DIT butterfly, the sequence of data appears differently. Eqn.

(C.27) shows an expanded form of the radix-4 DIF butterfly. Eighteen instructions are used to code the radix-4 DIF butterfly.

$$\begin{aligned}
 Ar' &= Ar + Br + (Dr + Cr) \\
 Ai' &= Ai + Bi + (Di + Ci) \\
 Cr' &= [(Ar - Br) - (Dr - Cr)]W_r^c + [(Ai - Bi) - (Di - Ci)]W_i^c \\
 Ci' &= [(Ai - Bi) - (Di - Ci)]W_r^c - [(Ar - Br) - (Dr - Cr)]W_i^c \\
 Br' &= [(Ar + Bi) - (Di + Cr)]W_r^b + [(Ai - Br) + (Dr - Ci)]W_i^b \\
 Bi' &= [(Ai - Br) + (Dr - Ci)]W_r^b - [(Ar + Bi) - (Di + Cr)]W_i^b \\
 Dr' &= [(Ar - Bi) + (Di - Cr)]W_r^d + [(Ai + Bi) - (Dr + Ci)]W_i^d \\
 Di' &= [(Ai + Br) - (Di + Cr)]W_r^d - [(Ar - Bi) + (Di - Ci)]W_i^d
 \end{aligned} \tag{C.26}$$

C.4.11 Inverse FFT

The Inverse Fast Fourier Transform (IFFT) is defined in Eqn. (C.27)

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi kn/N} \tag{C.27}$$

The differences between inverse FFTs and forward FFTs are in the scaling factor, N , and the conjugated twiddle factors. A common method of implementing the IFFT is to change the sign of the sine table values and use the FFT subroutine to get the IFFT. Alternatively, one can swap real and imaginary parts, use swapped inputs to the regular FFT program, and then divide every real and imaginary output by N . Eqn. (C.28) and Eqn. (C.29) show the equality. Eqn. (C.28) shows the inverse FFT.

$$(A_r + jA_i)(W_r + jW_i) = (A_r W_r - A_i W_i) + j(A_i W_r + A_r W_i) \tag{C.28}$$

When swapping real and imaginary parts at the input and using forward FFT twiddle factors, we have the relation shown in Eqn. (C.29).

$$(A_i + jA_r)(W_r - jW_i) = j(A_r W_r - A_i W_i) + (A_i W_r + A_r W_i) \tag{C.29}$$

Eqn. (C.29) shows that the real part of the IFFT is in the space used for imaginary memory in the forward FFT and the imaginary part of the IFFT is in the real part of the forward FFT.

C.5 OPTIMIZING PERFORMANCE OF THE FFT

“Optimization saves . . . 2067 instruction cycles which equals about 10% cycle time of the optimized code.”

C.5.1 Optimization

Judging the performance of any program requires the consideration of both its time and space complexity. There is always a trade-off between these two aspects. Time complexity indicates how fast an algorithm can be implemented on a specified microprocessor, while space complexity tells how much memory may be required. Optimization can either reduce memory requirement or minimize run-time of an algorithm. Since memory costs are continually decreasing, time optimization becomes more and more important.

One way to evaluate the time complexity of an algorithm is to compare its theoretical complexity, ideal implementation complexity, and practical complexity. Theoretical complexity refers to the number of additions and multiplications required by the given algorithm, independent of the microprocessor's architectures. This type of evaluating is only good for high-level comparison among algorithms and does not reflect the real performance of the algorithm on a given microprocessor. Not surprisingly, an algorithm that retains a lower theoretical complexity has a higher ideal implementation complexity. Ideal implementation complexity considers only the implementation of the core algorithm by the given microprocessor's instruction capabilities, such as available instruction type, addressing mode, parallel data move, etc. Ideal implementation complexity indicates the non-overhead performance of a given algorithm on a microprocessor, and always provides an optimistic estimation of an algorithm's performance. Practical complexity denotes the ideal implementation complexity plus the structure overhead of the microprocessor. (Structure overhead includes all required instructions not associated with the core algorithm.) Moving pointers, setting up DO loops, jumps to subroutines, and conditional jumps are typical structure overhead in microprocessors.

By distinguishing the different complexities, one can easily determine which microprocessor is competent for each aspect, and which instruction or address mode is critical to the specific algorithms. Also, chip designers may derive clues from the complexity analysis for determining which instruction or address mode should be added to the next revision. For example, the DSP96002 supports FMPY| |ADD| |SUB — an instruction with two parallel moves. The theoretical complexity of a radix-2 butterfly is four real multiplications and six additions or subtractions. Thus, the ideal implementation complexity of a radix-2 FFT on the DSP96002 is four instruction cycles. If each butterfly needs an average of 0.25 instructions to set up a pointer or DO loop, etc., the practical complexity of radix-2 is 4.25 instructions. The ratio of ideal implementation complexity to practical complexity reflects the efficiency of a microprocessor to perform a specific function. For example, the efficiency of the DSP96002 performing a radix-2 complex FFT could be:

$$\text{efficiency} = \frac{\text{ideal implementation complexity}}{\text{practical complexity}} = \frac{4}{4.25} = 0.94 \quad (\text{C.30})$$

In other words, the structure overhead for this particular example is about 6%. For FFTs implemented on programmable DSPs, the structure overhead should be between 3% and 15%. If a DSP has structure overhead higher than 15%, it cannot be called a DSP. If one claims a structure overhead lower than 3%, it is probably an application specific integrated circuit (ASIC).

C.5.2 Minimum Memory Requirement — In-Place Calculation

Although each radix-2 butterfly has two complex input data and two complex output data, calculation of the butterfly can be done by using only one memory set called in-place calculation. Memory requirements may be minimized by:

- **Reordering data into bit-reversed order.** This can be done in-place since data is interchanged by pairs, as seen in Eqn. (C.27). Thus, only $2N$ real data locations are required.
- **Reducing the size of the twiddle factor table** from N real locations to $N/2$ real locations for normal order input DIT FFT (see Reference 8). Notice that in normal order input DIT FFT the order that accesses the twiddle factor table is bit-reversal, i.e.

$$W_N^0, W_N^{(N/2)-1}, W_N^{N/4}, W_N^{N/8}, W_N^{(3N)/8}, \dots, W_N^{(N/4)-1}$$

$N/2$ complex numbers can be combined in pairs of two, which differ by a factor $W_N^{N/4} = -j$. In other words, the second twiddle factor in the pair can be obtained by multiplying $-j$ with the first twiddle factor. In fact, this optimization can be implemented with a minor modification to the previous butterfly core. All odd indexed groups will use negated, real and imaginary exchanged twiddle factors from the previous even indexed groups. Therefore, the number of groups in a pass is reduced to half of the previous one and the access time of twiddle factors is also reduced to half of the previous one.

- **Using a triple-nested DO-loop FFT** to minimize the program memory space (as seen in Figure C.17). Items 1 and 2 above save data memory space for the FFT calculation only.

C.5.3 Optimization for Faster Execution

Although the previously discussed program executes very efficiently, some applications may impose less stringent requirements on program memory size, but demand even faster execution. Faster execution can be obtained by further optimizing the previous algorithm. The following pages present several steps to achieve this optimization.

1. Since the first and second passes have trivial twiddle factors:

$$W_N^0 = 1, \text{ and } W_N^{N/4} = -j$$

it is common to combine the first and second passes as one radix-4 pass by calculating $N/4$ butterflies in the following equations.

$$\begin{aligned} Ar' &= Ar + Cr + Br + Dr \\ Br' &= Ar + Cr - (Br + Dr) \\ Cr' &= Ar - Cr + (Bi - Di) \\ Dr' &= Ar - Cr - (Bi - Di) \\ Bi' &= Ai + Ci - (Bi + Di) \\ Ci' &= Ai - Ci - (Br - Dr) \\ Ai' &= Ai + Ci + Bi + Di \\ Di' &= Ai - Ci + (Br - Dr) \end{aligned} \tag{C.31}$$

Notice that there are eight additions and eight subtractions in Eqn. (C.23). A DSP that has a multiplication and accumulation instruction with one or two parallel moves (type A DSP) may take at least sixteen instructions to do Eqn. (C.23). A DSP that has a FMPY | ADD | SUB instruction with two parallel data moves (type B DSP) can do Eqn. (C.23) in eight instructions. After combining the first two trivial passes as a radix-4 pass, the number of instructions required in the radix-2 DIT complex FFT becomes:

$$(\text{TRIV} \times N/4) + [(m - 2) \times N/2 \times \text{BFLY}]$$

where: TRIV is the number of instructions necessary to perform a trivial butterfly

Theoretically, for the DSP56001/2, the DSP96002, and the DSP56156, TRIV may be 16, 8, and 16 instruction cycles, respectively. Therefore, a 1024-point complex FFT on the DSP96002 can be done in $(8 \times 256) + (8 \times 512 \times 4) = 18,432$ instruction cycles. This is a lower boundary of the radix-2 complex FFT. In fact, TRIV is 17, 8, and 22 on the DSP56001, the DSP96002, and the DSP56156, respectively. Cycle time of the FFT can be reduced further by exploring the simple relations among the remaining passes.

2. Trivial twiddle factors exist in the remaining passes as well. Special butterflies can take advantage of those simple relations. There are two types of trivial twiddle factors:

$$\text{Type I } W_N^0 = 1, W_N^{N/4} = -j$$

$$\text{Type II } W_N^{N/8} = -W_N^{(3N)/8} = 0.707 - j0.707$$

Type I trivial factors don't involve multiplications as already shown in Eqn. (C.31). To utilize these simple relations in the remaining passes, different butterflies must be inserted in one pass. This change results in longer program code and some structure overhead, such as updating address registers, different DO loops, and modulo addressing.

Type II trivial factors are not really trivial for either type A or type B DSPs. Type II trivial factors reduce the theoretical complexity of a radix-2 butterfly to two real multiplications and six real additions or subtractions. With only one adder on type A DSPs, six instructions are required as before. The ideal implementation complexity could be 3 for type B DSPs, but unfortunately each radix-2 butterfly deals with four real inputs and four real outputs. Type B DSPs have only two parallel data moves, and each radix-2 butterfly still takes at least four instruction cycles for type II trivial factors. *The type II trivial factor issue is addressed here because this is probably the last chance for further optimizing radix-2 FFTs.*

Each group in the last pass consisted of a single butterfly. A triple nested DO loop is thus no longer required in this pass: it can be split and handled by a single DO loop.

3. Another alternative is to combine the last two passes into one radix-4 pass. Since each butterfly in the last pass requires a different twiddle factor, one instruction to fetch a twiddle factor must be appended in the butterfly core. The same fetch occurred in the second to last pass in every two butterflies. Combining four radix-2 butterflies into one radix-4 butterfly may save four multiplications but a special twiddle factor table has to be created for the radix-4 butterfly.
4. For longer FFTs (>256 points), internal memory in the DSP56001/DSP56002 is not sufficient to contain the complete data set. Consequently, the butterflies execute more slowly when the processor needs to fetch a data value in external X and in external Y memory in the same instruction cycle. This causes the instruction cycle to be "stretched," resulting in slower execution time. Through intelligent memory usage, however, this effect can be minimized. In a further optimized routine (see **Section C.10**), the first two passes are combined into a single pass. Next, separate 256-point FFTs are computed, whereby the data is moved into internal memory, and the results are not moved to external memory until the final pass. This process avoids the stretching of the instruction cycle on the middle passes, and makes optimal use of the available internal memory.

With these optimizations, a significantly faster routine is obtained. For instance, a 1024-point optimized complex FFT routine is available for DSP96002 which executes in 0.94 ms at 40MHz clock (see Fully Optimized Complex FFT in **Section C.10**). A fully optimized complex FFT routine for DSP56001/2 is also listed in **Section C.10** (CFFT56.ASM). 0.704ms is needed to calculate a 512-point complex FFT at 40 MHz clock, which is 8.7% faster than an optimized complex FFT. For more benchmarks see **SECTION C.8**. Note, however, that "straight-line" code always results in longer programs.

C.5.4 Example of Optimization

C.5.4.1 Fully Optimized Complex FFT for the DSP56001/2 Program CFFT56.ASM in **Section C.10** is a good example of optimizing complex FFTs on the DSP56001/2 for fast execution time. Figure C.26 shows passes, groups and butterflies for a 512-point complex FFT. There is a total of 9 passes. The number of groups, in each pass doubles from pass to pass, while the number of butterflies in each group halves from pass to pass. Each pass has the same number of butterflies, i.e. $N/2=256$ butterflies.

CFFT56.ASM takes advantage of the trivial twiddle factors in all the passes. Note that passes 0 and 1 can be done by simple radix-4 butterflies. A radix-4 butterfly has been coded by 17 instructions, which is the best case on the DSP56001/2. The parallel data move in this radix-4 butterfly has been deliberately arranged to avoid a dual

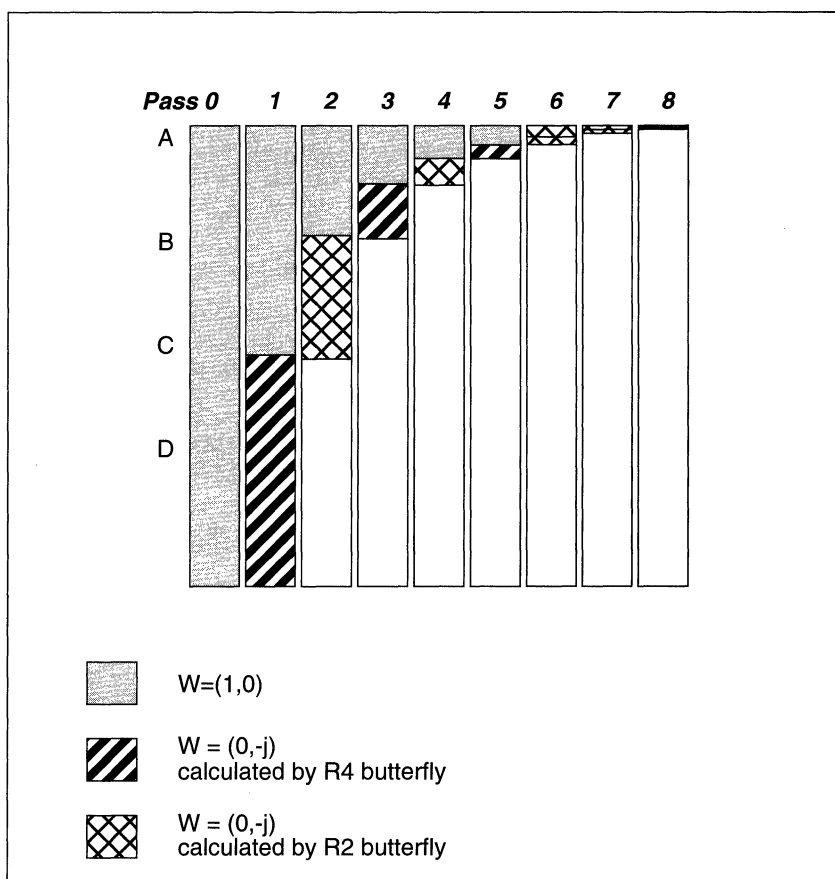


Fig. C.26 Trivial twiddle factors in a 12-point complex radix-2 DIT FFT. The butterflies in highlighted groups can be calculated without multiplications. A, B, C, and D are radix-4 butterfly pointers.

data move involving external memory, although the first and next to last instructions may result in cycle stretch in some cases. Since half of the 512 data are in external memory, one instruction cycle is stretched, and 18 instruction cycles are used for a 512-point complex FFT. This equals 4.5 instruction cycles per radix-2 butterfly. The same radix-4 butterflies are also applied to passes 2, 3, 4, and 5. Note that in Figure C.26, the groups highlighted by cross lines are trivial butterflies too, and are not covered by the simple radix-4 butterflies. These data points are calculated by 5-instruction radix-2 butterflies. As shown in Figure C.26, each pass has 256 radix-2 butterflies and the first seven passes have 860 trivial butterflies. 772 of these radix-2 butterflies require 4.5 instruction cycles (simple radix-4 butterflies) while 88 of them require 5 instruction cycles. Therefore, the total cycle time for trivial butterflies is $772 \times 4.5 + 88 \times 5 = 3,914$ which means a savings of $860 \times 6 - 3,914 = 1,246$ cycles when compared to a non-optimization case. For program simplicity, the above calculation does not utilize the trivial butterflies in passes 7 and 8.

CFFT56.ASM uses $N/2$ real twiddle factors. This scheme reduces the data memory requirement and also reduces the structure overhead on group DO loops, because the group number in each pass changes to half of the previous scheme.

CFFT56.ASM fully utilizes internal memory to avoid cycle stretch when the DSP56001/2 accesses two data. A 512-point complex FFT is divided into two 256-point parts. The first 256-point part remains in internal memory until the last pass. The second 256-point data loads into internal memory after the first pass and stays there until the last pass.

The last two passes are implemented by two separate single loops to avoid the penalty of DO loop set-up. Each group has four radix-2 butterflies in the next-to-last pass, and two in the last pass. If group DO loop is still used, then each butterfly may take 6.75 and 7.5 cycles in the next-to-last pass and the last pass, respectively. The cycles saved from the separated DO loops are $256 \times 6.75 + 256 \times 7.5 - 512 \times 6 = 576$.

C.5.4.2 Fully Optimized Complex FFT for the DSP96002 Section C.10 presents a fully optimized program for 1024-point complex input FFT for the DSP96002. Like the fully optimized program for the DSP56001/2, this program takes advantage of trivial twiddle factors in all of the passes as follows:

- ☛ Naturally, the first and second passes are combined into a radix-4 pass with each radix-4 butterfly requiring 8 instruction cycles. This is equal to 2 instruction cycles per radix-2 butterfly.
- ☛ All trivial butterflies in the middle passes are calculated by a separate routine.
- ☛ Each pass is written in a separate section to reduce the DO loop overhead. To reduce the program length, the special radix-4 and normal radix-2 butterflies are programmed in subroutines. Only two-nested DO loops are used for each pass.
- ☛ The last two passes are also combined into a radix-4 pass. After the combination, the number of instruction cycles per radix-2 butterfly is decreased from 5 to 4.25 instruction cycles. Because radix-4 butterflies are used in the last two passes, an extra set of 256 complex twiddle factors must be present in the external memory. These twiddle factors are generated off-line by MATHLAB software.

The fully optimized 1024-point complex FFT uses 18891 instruction cycles; while the optimized 1024-point complex FFT program (seen on the Motorola DSP bulletin board; Dr. BuB) uses 20958 instruction cycles. Optimization saves $20,958 - 18,891 = 2,067$ instruction cycles which equals about 10% cycle time of the optimized code. Also note that the fully optimized code only works with fixed data length.

C.6 REAL-VALUED INPUT FFT ALGORITHM

"Data acquisition on the DSP96002 is truly parallel with CPU instruction execution."

A real-valued input FFT is a special case of the complex FFT where all imaginary components in the input are zero. Under this condition, input sequence is real, and the time sequence has a symmetric Fourier transform in the frequency domain. Only half of the frequency sequence needs to be computed for real-valued input FFTs or real FFTs. Recall the definition of the DFT:

$$X(k) = \sum_{r=0}^{N-1} x(r)e^{-j(2\pi rk)/N} \quad k = 0, 1, \dots, N-1 \quad (C.32)$$

If $x(r)$ is real,

$$X^*(-k) = \sum_{r=0}^{N-1} x(r)e^{j(2\pi rk)/N} = \sum_{r=0}^{N-1} x(r)e^{-j(2\pi rk)/N} = X(k) \quad (C.33)$$

and

$$X^*(N-k) = \sum_{r=0}^{N-1} x^*(r)e^{(-j)(2\pi r(N-k))/N} = \sum_{r=0}^{N-1} x(r)e^{-j(2\pi rk)/N} = X(k) \quad (C.34)$$

C.6.1 Real-Valued Input FFT Algorithm 1

C.6.1.1 Bergland Algorithm This algorithm was developed by Glenn D. Bergland in 1968 (see Reference 15). To derive this algorithm, we assume that readers are familiar with the Cooley-Tukey radix-2 DIT complex FFT shown in Figure C.10.

Bergland's algorithm is based on the observation of the symmetry of the FFT to the real input, $X_N(k) = X_N^*(N-k)$. Calculating the second half of the FFT is not necessary. By checking for redundancy in the Cooley-Tukey radix-2 decimation in time complex FFT when input is a real sequence, one may discover that when the twiddle factors equal $W(N/4) = -j$, only a negation and a re-labeling need be performed. This so-called re-labeling simply exchanges real and imaginary data indexed by address registers. All odd index outputs in Figure C.10 are the second half of the

transform, which can be obtained from the symmetry. Bergland's algorithm uses those memory locations for storing imaginary values. A direct map from the Cooley-Tukey algorithm to Bergland's algorithm is diagrammatically shown in Figure C.27. Note that all inputs are real and all intermediate results are stored in N and *only* N locations. The calculation can be done in-place; however, the indices of each butterfly outputs are not in bit-reversed order as in the Cooley-Tukey algorithm. The following discussion refers to this order as the Bergland order.

The twiddle factors appear to be in the Bergland order also, as shown Figure C.27,

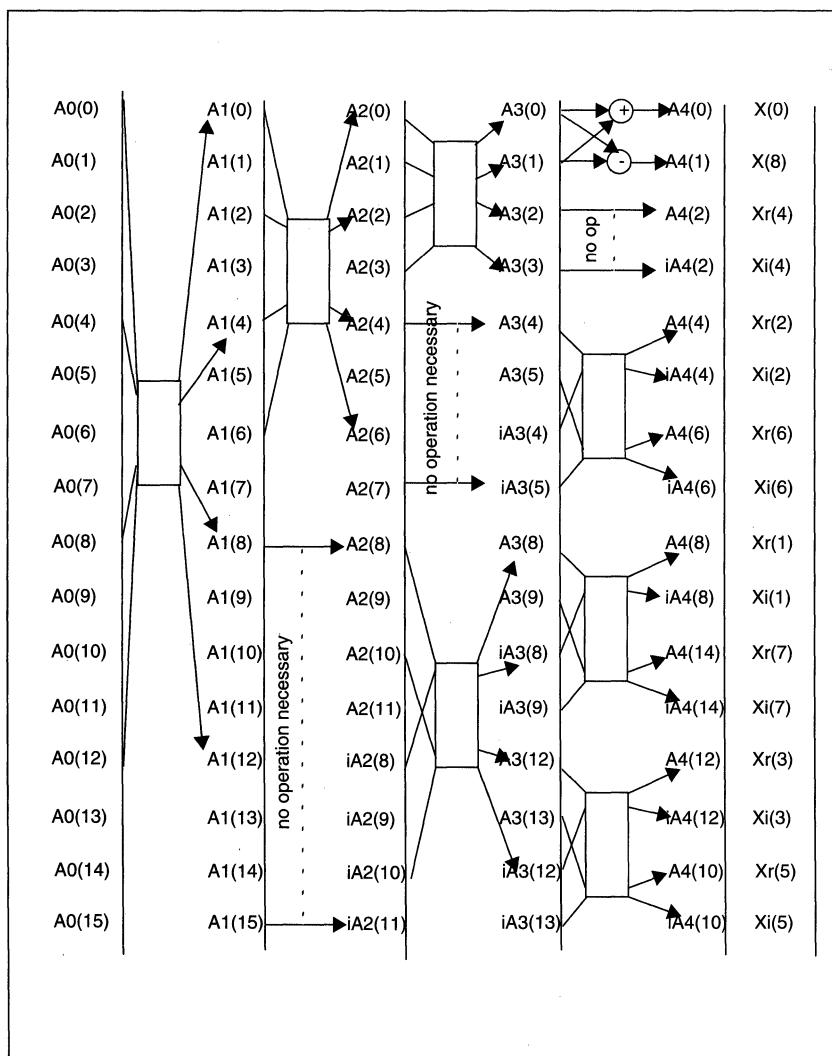


Fig. C.27 Non-redundancy calculation of the Cooley-Tukey radix-2 DIT FFT with real inputs

if more than 16 points of real FFT are carried out. The next section explains how to convert a normal order of twiddle factors to the Bergland order and how to convert the Bergland ordered outputs to normal order. The only operation performed for multiplying by $-j$ is a re-labeling of half of the current outputs as imaginary inputs for the next stage. Thus, in Figure C.15 all butterflies, except one with W^0 , have the crossed inputs to the butterfly, even though the butterfly in each group is identical. An additional benefit of 'no operation' is the reduction of the number of passes, $\log_2(N)-1$, except for one addition and one subtraction. The final algorithm is shown in Figure C.28.

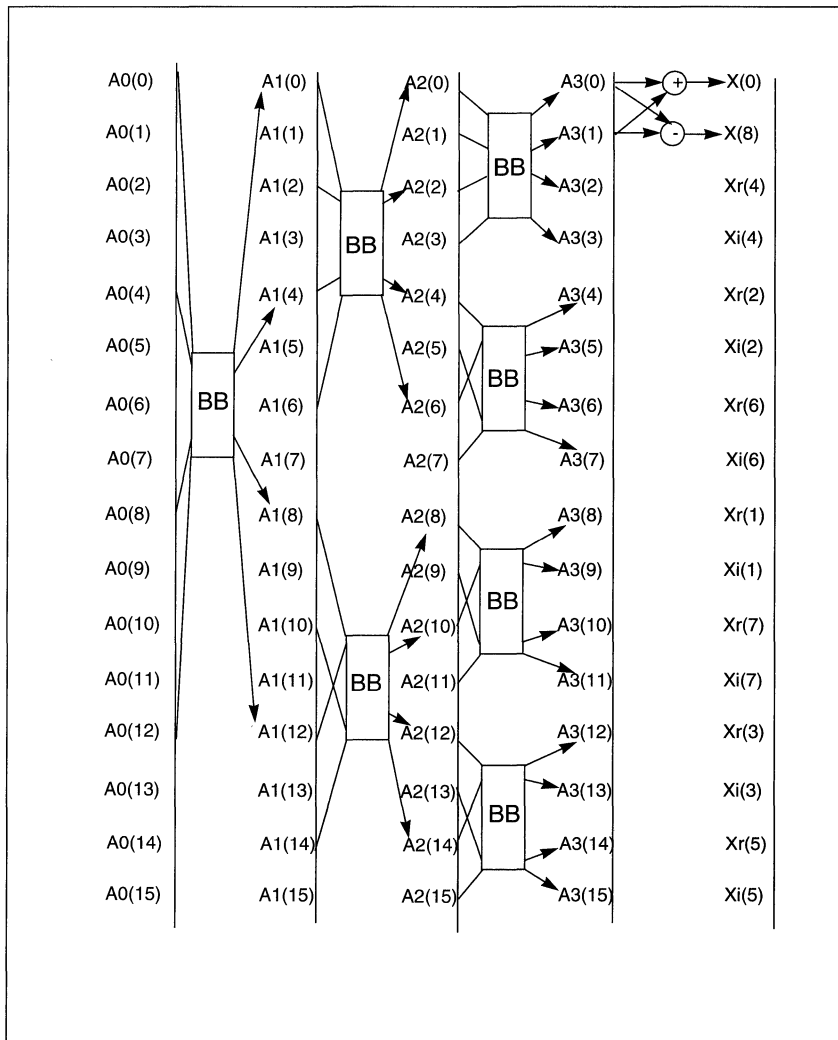


Fig. C.28 Bergland algorithm has only $\log_2(N)-1$ passes and one more addition and subtraction

The Bergland butterfly differs from the Cooley-Tukey butterfly simply in that the Bergland requires two more conjugate operations, which are done by re-labeling (see Figure C.29). Essentially, the number of arithmetic operations required by both algorithms is the same. Although re-labeling can be implemented in parallel with other arithmetic operations without consuming instruction cycle time, it does require a data move. This extra traffic may have an impact on the implementation later on. Figure C.29 depicts the Bergland butterfly. Butterfly (a) is a simplified version of (b) since no complex multiplication is carried out when $w=1$. Note that the inputs in (b) have been re-labeled to reflect a multiplying $-j$ operation. To calculate the butterfly (a) two additions and two subtractions are needed along with four real multiplications, three real additions, and three real subtractions.

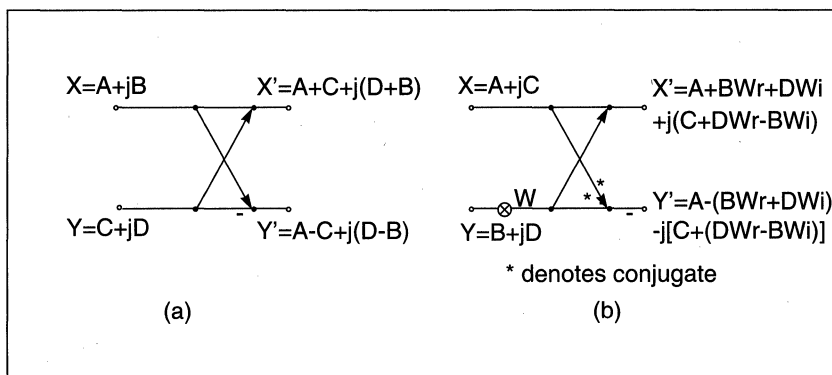


Fig. C.29 (a) Butterfly of Bergland Algorithm with $W = 1$, (b) Butterfly of Bergland Algorithm with $W \neq 1$

C.6.1.2 Reordering The output order of the Bergland algorithm is slightly different than the bit-reversed order, and the twiddle factor required in the calculation is also in Bergland order. To get this special order, one may use the following algorithm for doubling the length of each number sequence:

1. Multiply the second entry of the sequence by two, and make this product the second entry of the new sequence
2. Subtract each nonzero entry of the sequence from twice the product formed in step 1 (these differences form the rest of the even entries of the new sequence)
3. Take the odd entries of the new sequence as the numbers of the original sequence

The algorithm in Figure C.29 can be translated to the C language code in Figure C.30.

Also note that the size of the twiddle factors required in Bergland FFT is $N/4$, while the size of the output data is $N/2$. Two tables must be generated before the FFT computation.

Fig. C.30 C language code that generates Bergland order tables

```

void
bildberg(bergtabl,buf_size)
short *bergtabl,buf_size;

{
    register int i,j,k;
    i = buf_size / 4;
    k = 4;

    bergtabl[0] = 0;                /* seed values for start */
    bergtabl[i] = 2;
    bergtabl[2*i] = 1;
    bergtabl[3*i] = 3;

    while(i>1)
    {
        i = i/2;                    /* increments drop by half */
        k = k*2;                    /* new sequence size doubles */

        bergtabl[i] = k / 2;
        for (j=i+i; j<buf_size; j = j+i+i)
            bergtabl[j+i] = k - bergtabl[j];
    }
}

```

C.6.1.3 Performance Estimation For $N=2^m$, it has been shown that the pass or stage number in Bergland algorithm is $\log_2(N)-1=m-1$. In each pass there is one (and only one) type (a) butterfly group. The Bergland algorithm takes four points in and four points out. The number of butterflies in each pass is $N/4$. Each butterfly uses four multiplications, three additions, and three subtractions, except that the type (a) butterfly uses only two additions or two subtractions. For $N=2^m$, Bergland algorithm may need

$$4 \times N/4 + \sum_{i=2}^{m-1} [4 + BB(2^{i-1} - 1)]N/(2^{i+1}) \quad (C.35)$$

instruction cycles to perform an N -point real FFT, where BB is the number of instructions for the Bergland butterfly. Theoretically, for the DSP96002 and the DSP56001, BB should be 4 and 6, respectively. If the normal order output is desired, then converting Bergland order data to the normal order data must be included in the performance estimation. At least two more instructions have to be added to the last pass for accessing the Bergland order table. The performance of the Bergland algorithm including unscrambling could be:

$$N + \sum_{i=2}^{m-1} [4 + BB(2^{i-1} - 1)](N/2^{i+1}) + (N/2) - 1 \quad (C.36)$$

Eventually, the real performance of an FFT is determined by the architecture of the DSP on which the FFT runs. As described in **SECTION C.4.4**, the actual performance of the FFT is determined by the number of data paths, the number of registers, the instruction type, the cycle time of DO loop, and the memory organization. In other

words, a good or relatively low complexity algorithm may not generate good performance if the microprocessor's architecture does not provide hardware support for that algorithm. Due to the memory structure and instruction type, the number of instructions for a Bergland butterfly, (BB), actually are 5 and 7 on the DSP96002 and the DSP56001, respectively. (See programs RFFT96B.ASM and RFFT56B.ASM in **Section C.10**.) Due to this compromise in the implementation, the next algorithm may be preferable because of the number of instructions.

C.6.2 Real-Valued Input FFT Algorithm 2

The second algorithm treats an N real-valued input array as an $N/2$ complex array, without extra zeros. Then, an $N/2$ complex FFT is performed. The trick is to separate the transformation of the complex sequence into two complex sequences, then to obtain the transformation of the real-valued input array.

C.6.2.1 Separating Two Real FFT from One Complex FFT If a real-valued input array is $z(n)$, its transform $Z(k)$ has an even real part and an odd imaginary part. If $z(n)$ is packed in such a way that all even index data is in $x(n)$ and all odd index data is in $y(n)$, then,

$$\begin{aligned}
 Z(k) &= \text{DFT}[z(n)] = \text{DFT}[x(n) + jy(n)] \\
 &= (\text{DFT}[x(n)] + j\text{DFT}[y(n)]) \\
 &= (X_r(k) + jX_i(k) + j[Y_r(k) + jY_i(k)]) \\
 &= ([X_r(k) - Y_i(k)] + j[X_i(k) + Y_r(k)])
 \end{aligned} \tag{C.37}$$

Eqn. (C.37) shows that the DFT of a complex time sequence $z(n)$ can be represented by the DFTs of two real time sequences $x(n)$ and $y(n)$, because the DFT is a linear transform.

Also the second half of $z(n)$ can be represented by the DFT of $x(n)$ and $y(n)$

$$Z(N-k) = [X_r(k) + Y_i(k)] - j[X_i(k) - Y_r(k)] \tag{C.38}$$

The goal of the derivation is to find out how to construct the DFT of two real time sequences from the DFT of a complex sequence. By combining Eqn. (C.37) and Eqn. (C.38), it shows:

$$\begin{aligned}
 \text{DFT}[x(n)] &= X_r(k) + jX_i(k) \\
 &= \{[Z_r(k) + Z_r(N-k)] + j[Z_i(k) - Z_i(N-k)]\}/2
 \end{aligned} \tag{C.39}$$

$$\begin{aligned}
 \text{DFT}[y(n)] &= Y_r(k) + jY_i(k) \\
 &= \{[Z_i(N-k) + Z_i(k)] + j[Z_r(N-k) - Z_r(k)]\}/2
 \end{aligned}$$

where: $k = 0, 1, \dots, N/2$

According to Eqn. (C.39), two DFTs of two real time sequences can be rebuilt from one complex DFT. This split operation, which separates two DFTs from one, paves the way for the calculation of N real input DFTs done by an $N/2$ complex DFT.

C.6.2.2 Rebuilding the DFT of a Real Sequence from Two DFTs From the previous discussion, DFTs of two real sequences can be constructed from one complex DFT. In this section, we investigate how to rebuild the DFT of a real sequence from two DFTs. To understand this point, recall Eqn. (C.16). It can be rewritten as:

$$F(k) = X(k) + W_N^k Y(k) \quad k = 0, 1, \dots, N-1 \quad (C.40)$$

where:

$$X(k) = \sum_{r=0}^{N/2-1} x(2r) W_{N/2}^{rk}$$

$$Y(k) = \sum_{r=0}^{N/2-1} x(2r+1) W_{N/2}^{rk}$$

Note that $X(k)$ is the DFT of the even index sequence and $Y(k)$ is the DFT of the odd index sequence. $X(k)$ and $Y(k)$ in Eqn. (C.40) can be determined from Eqn. (C.39). Furthermore, $F(k)$, the DFT of a real sequence with N points, can be found according to Eqn. (C.40). Combining Eqn. (C.39) and Eqn. (C.40), we obtain the final equation Eqn. (C.41).

$$F(k) = \frac{[Z(k) + Z^*(N/2 - k)]}{2} - j \frac{[Z(k) - Z^*(N/2 - k)]}{2} W_N^{rk} \quad (C.41)$$

where: $k = 0, 1, \dots, (N/2)-1$,

N = Number of real inputs

Notice that:

- ☞ Only 0 to $N/2-1$ points are saved by the algorithm.
- ☞ The values $F(0)$ and $F(N/2)$ are real and independent, to obtain entire spectrum, $F(N/2)$ in the imaginary part of $F(0)$.
- ☞ The twiddle factors in the DFT and split complex multiplication have different resolutions. In the DFT, the period of W is $N/2$; in the split complex multiplication, the period of W is N , though the same number of points ($N/2$) are needed in both cases. This means the algorithm may use more memory space for twiddle factors.

Eqn. (C.41) can be decomposed further to a real multiplication format that can be implemented on DSPs.

$$\begin{aligned}
H1_r &= (A_r + B_r)/2 \\
H1_i &= (A_i - B_i)/2 \\
H2_r &= (A_i + B_i)/2 \\
H2_i &= (B_r - A_r)/2 \\
A'_r &= H1_r + (W_r H2_r - W_i H2_i) \\
B'_r &= H1_r - (W_r H2_r - W_i H2_i) \\
A'_i &= H1_i + (W_i H2_r - W_r H2_i) \\
B'_i &= -(H1_i) + (W_i H2_r - W_r H2_i)
\end{aligned} \tag{C.42}$$

where:

$$\begin{aligned}
W_r &= \cos((2\pi k)/N) \\
W_i &= -\sin((2\pi k)/N)
\end{aligned}$$

and

$$\begin{aligned}
A_r &= \text{real}Z(k) & k=0, \dots, (N/4-1) \\
B_r &= \text{real}Z(N-k) & k=0, \dots, (N/4-1) \\
A_i &= \text{imag}Z(k) & k=0, \dots, (N/4-1) \\
B_i &= \text{imag}Z(N-k) & k=0, \dots, (N/4-1)
\end{aligned}$$

C.6.2.3 Performance Estimation In the following paragraph, we will discuss the computational complexity of Eqn. (C.41) and the implementation constraints on the architecture of Motorola's DSP. For detailed implementation, please refer to the programs RFFT96.ASM and RFFT56.ASM in **Section C.10**.

Eight multiplications, five additions, and five subtractions are needed to implement Eqn. (C.41). The minimum requirement for this calculation is eight instructions if one multiplier and the MPY | ADD | SUB is available on the given DSP. Note that there are four special multiplications, and the multiplicands are 1/2 in the calculations of $H1_r$, $H1_i$, $H2_r$, and $H2_i$.

On the DSP56001, the divide-by-2 operation can be automatically implemented by a "scaling down" mode when data moves from the ALU accumulator (A or B) to the X or Y data bus occur. The cost of implementing the division operation, of course, is that one instruction has to be used to turn on the scaling down bit in the Status Register. Apparently, only four multiplications are needed on the DSP56001. But one may find that when the scaling down mode is on, all output data from the accumulator (A or B) to X or Y memory is also divided by 2. Thus, the scaling down mode has to be turned off before data is output to the X or Y memory.

The scaling bit control instructions on the DSP56001 do not allow parallel data moves or any other operations. If the DSP is in the scaling mode, a total of twelve instructions are needed: four MAC instructions, two toggling scaling bit instructions, and six more ADD or SUB instructions. In practice, see program RFFT56.ASM in **Section C.10**, where the scaling mode is never turned on because scaling must be done if block floating point is not used. Therefore, the output of the program RFFT56.ASM is twice as large as true values. Ten instruction cycles is the minimum requirement. In practice, one instruction in the loop for data saving is included.

On the DSP96002, since the FMPY || ADD || SUB instruction is available, eight instructions are enough to perform a computation such as Eqn. (C.41). In **Section C.10** more details about implementation such as memory map, program length, twiddle factors, and data size are presented.

The overall performance of the algorithm is determined by the time required to calculate an $N/2$ complex FFT plus the time for separating manipulations.

$$CFFT(N/2) + S \times N/4 \quad (C.43)$$

where: $S = 11$ for the DSP56001

$S = 8$ for the DSP96002

C.6.3 Real-Valued Input FFT Algorithm 3

In most practical situations, the data to be analyzed by the FFT is real and is usually obtained from a single analog-to-digital (A/D) converter. This knowledge can be exploited in several ways to increase the speed of the FFT calculation even further:

1. Since the input data is real, there is no need to multiply, add, or subtract the imaginary parts.
2. Use can be made of symmetries within the FFT:

$$X_N(k) = X_N^*(N - k) \quad (C.44)$$

When $x(nT)$ is real, $*$ denotes complex conjugate.

Clearly, not all of the frequency points need to be calculated, as many of them can be obtained by taking a simple complex conjugate of other, previously computed points. Taking a complex conjugate can be easily achieved by moving the same values to different memory locations, after taking the negative of the value which goes to Y memory (imaginary part). Figure C.31 shows the procedure for a 16-point, real FFT in greater detail. A real-input FFT routine is available for the DSP56001/2, which executes in 1.01 ms using a 40-MHz clock. This also includes the amount of time necessary to bring in 1024 sampled data points from an external A/D converter. Because of the fast interrupt capability of the DSP56001/2, data sampling creates very little overhead. As a result, the maximum sampling rate at which a 1024-point real FFT can be executed equals:

$$F_{smax} = \frac{1024}{1.01 \times 10^{-3}} = 1.014(\text{MHz})$$

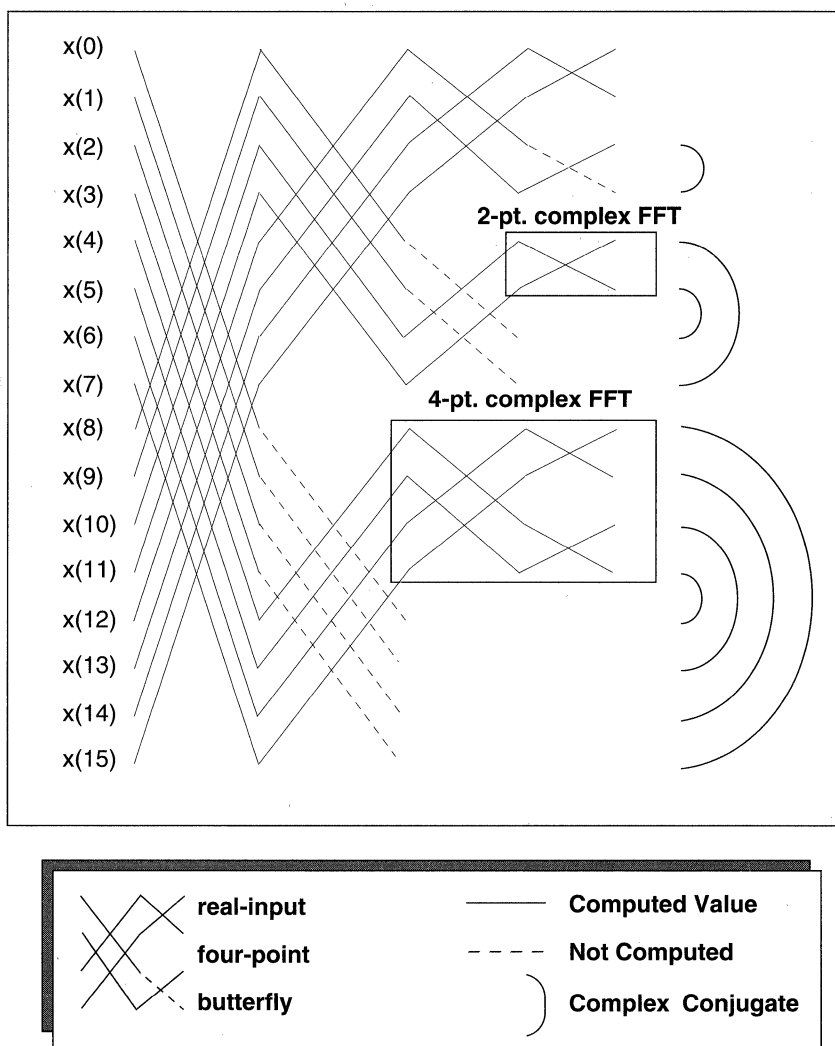


Fig. C.31 Computation of the Real-Input, DIT FFT

Comparing this with the sampling rate of 3.3 kHz mentioned in **SECTION C.3.1 Motivation**, a more than 300-fold improvement is obtained by carefully optimizing the Fourier transform algorithm!

C.6.4 The Goertzel Algorithm

Previous FFT algorithms compute all or half of the frequency points in the range equaling half of the sampling rate. For some applications, such as single frequency

detection, only one or several frequency points are of interest. Using FFT to find these frequencies is no longer cost effective in the sense of computational complexity.

The Goertzel algorithm (see Reference 3) can be implemented by a second order IIR filter for each DFT coefficient. The transfer function for the IIR filter is:

$$H_k(Z) = \frac{1 - W_N^k Z^{-1}}{1 - 2\cos(2\pi k/N)Z^{-1} + Z^{-2}} \quad (\text{C.45})$$

where: $W_N^k = e^{-2\pi kj/N}$

N = the length of input sequence, which depends on the resolution of two consecutive frequencies to be detected

k = the index of DFT coefficient

Also note that only three real coefficients are required in the IIR filter. Naturally, the IIR filter recursively works on input samples and output results, so no input data buffer is needed; and only two memory locations are used for storing internal states of the IIR filter. Figure C.32 shows an implementation of the Goertzel algorithm by a second order IIR filter. In contrast, an IIR filter calculates every output corresponding to every input. In the Goertzel algorithm, only one DFT coefficient $X(k)$ is needed, and $X(k) = y_k(N)$. In other words, the complex multiplication is carried out only once in an entire DFT calculation. In frequency detection, only the power of magnitude of the DFT coefficient is needed. This observation may simplify the computation even more.

Fig. C.32 DSP56001 assembly code that calculates energy of DFT coefficients by single parameter

```

;Goertzel algorithm to calculate energy of DFT coefficient
:
:
:
data      equ      $100
COEF      equ      $123456
LOOP      equ      256

      org p:$40
      move      #data,r0          ;r0 -> input data
      clr a      #0,b            ;I(n-1)=0,I(n-2)=0
      move      #COEF,y0         ;y0=cos(2pik/N)
      do #LOOP,_END_GOERT
          neg b y:(r0)+,a a,x1    ;x1=I(n-1),b=-I(n-2),a=x[i]/2
          macr y0,x1,a x1,y1      ;a=x[i]/2 + I(n-1)*COEF,y1=I(n-1)
          addl b,a x1,b           ;a=x[i] + 2*I(n-1)*COEF - I(n-2),b=I(n-1)
      _END_GOERT
      mpy -y0,x1,a a,x0          ;a=-con(2pik/N)I(n),x0=I(n)
      mpy x1,y1,b                ;b=I(n-1)^2
      mac x0,x0,b a,y0           ;b=I(n)^2+I(n-1)^2
      mpy x1,y0,a                ;a= -con(2pik/N)I(n)I(n-1)
      addl b,a                    ;a= power of magnitude of DFT

```

From Figure C.32, the last output of the IIR filter is:

$$y_k(N) = |I(N) - W_N^k I(N-1)| \quad (\text{C.46})$$

The power of magnitude of the DFT coefficient is easy to show:

$$|y_k(N)|^2 = |^2(N) - 2\cos(2\pi k/N)|N|(N-1) + |^2(N-1) \quad (\text{C.47})$$

Hence, only one real coefficient is required to compute the energy of the signal. Figure C.32 shows the DSP56001 assembly language code used to detect the energy of a frequency specified by the Goertzel algorithm. The recursive part of the IIR filter is effectively implemented by three instructions. The total instruction cycles for an N-point input sequence is $3N+8$. Only one coefficient $\cos(2\pi k/N)$ is stored in the on-chip memory and two more memory locations are used to store internal states $I(N)$ and $I(N-1)$.

C.6.5 Real-Time Data Acquisition on Motorola DSPs

A very important feature of a DSP is its capability to carry data in and out in a deterministic amount of time without interfering with the CPU core operations. "Real-time FFT" refers to the sampled data from an A/D converter or other devices that is stored in a buffer. Once this buffer is full, the DSP starts the FFT program execution. In the mean time, the DSP grabs the sampled data and puts it into another buffer. Whichever finishes first, (the FFT program execution or data acquisition), has to wait for the other one to finish its task. Thus, two data buffers, plus synchronization between the program execution and data acquisition, are required to implement the real-time FFT. This is also called double buffering. The following sections present the I/O peripherals on the DSP56001/2 and the DSP96002, and show examples of how to set up these peripherals for real-time data acquisition.

C.6.5.1 Fast Interrupt on DSP56001 for Real-Time FFT Data Acquisition Figure C.33 shows a scheme for double buffering. Two memory spaces are exclusively assigned to an FFT program. The FFT program will not start until one of two buffers is full. The loaded buffer will not be loaded with data again unless the FFT has finished its execution on the buffer.

The double buffering is implemented by the fast interrupt on the DSP56001/2 (see Reference 1). The data received by peripherals such as the SSI or Host Interface

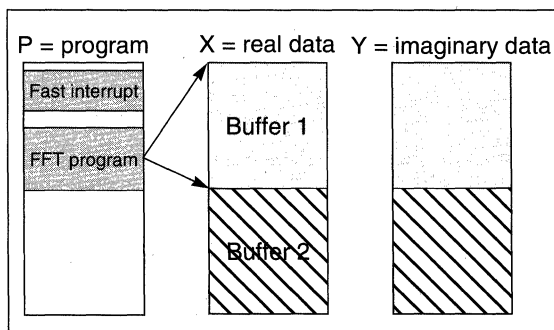


Fig. C.33 Double buffering input data so that data input can work with the FFT program concurrently

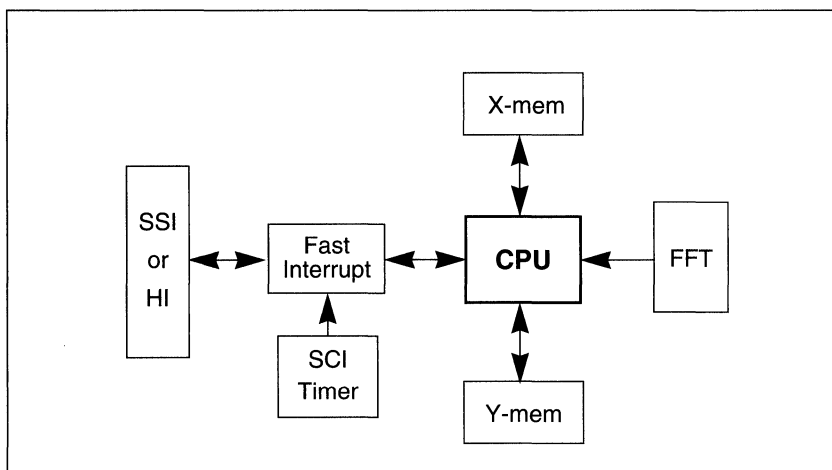


Fig. C.34 Block diagram of the double buffering technique. SSI/HI fast interrupt has higher priority than the MAIN or FFT program. The pointer of buffer is checked by SCI timer interrupt which has highest interrupt priority. The interval of the timer interrupt is set according to data length so that the buffer pointer can be updated accordingly.

(HI) on the DSP56001/2 will be moved into the internal memory by the fast interrupt. The fast interrupt needs only two instruction cycles to move one received data word from a peripheral to a specified memory location without changing the program flow in the CPU.

The data generation rate is actually much slower than the FFT speed. For example, to generate a set of 1024-point data at 44.1 kHz sampling rate could take $1/44100 \times 1024 = 23.2(\text{ms})$ while a 1024-point real FFT only takes about 1ms at 40 MHz clock on the DSP56001/2. For this reason, the SSI or HI interrupt as shown in Figure C.34 has been assigned higher priority than the FFT program so that every piece of data received can be sent to internal memory via fast interrupt on the DSP56001/2. The buffer pointer keeps growing by SSI/HI data moves and is being checked by the SCI timer interrupt. Once the buffer is full, the FFT program starts and proceeds to move the buffer pointer to the next buffer so that SSI/HI fast interrupt works with the CPU concurrently.

C.6.5.2 Real-Time Data Acquisition on DSP96002 The same double buffering technique used on the DSP56001 for real-time data acquisition is also applicable on the DSP96002. Data acquisition on the DSP96002 is truly parallel with CPU instruction execution. Recall the DSP96002 architectural block diagram in Figure C.18. The double buffering technique guarantees that the two DMA channels directly connected to the internal memory support parallel data access without stretching an instruction cycle if the CPU core and the DMA controller access different internal memory locations.

C.7 TWO DIMENSIONAL FOURIER AND COSINE TRANSFORMS

"To implement Eqn. (C.48), a two dimensional time sequence is decomposed according to its row or column."

Two dimensional Fourier transforms are widely used in image processing, image analysis, and video compression. Because the fast discrete cosine transform features high energy compaction and low implementing complexity, it is becoming more and more important in image and video compression.

C.7.1 Two Dimensional FFTs on the DSP96002

Two dimensional FFTs are simply an extension of one dimensional FFTs, as is shown by:

$$F(i, k) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(m, n) e^{-j(2\pi mi)/N} e^{-j(2\pi nk)/N} \quad (C.48)$$

where: $i = 0, 1, \dots, N-1$

$k = 0, 1, \dots, N-1$

To implement Eqn. (C.48), a two dimensional time sequence is decomposed according to its row or column. Eqn. (C.48) can be rewritten in Eqn. (C.49).

$$F(i, k) = \sum_{m=0}^{N-1} \left(\sum_{n=0}^{N-1} x(m, n) e^{-j(2\pi mi)/N} \right) e^{-j(2\pi nk)/N} \quad (C.49)$$

The one dimensional FFT code discussed in **SECTION C.4** can be used in this extension. The code included on the Motorola DSP bulletin board (2DFFT.asm) implements the two dimensional DIT FFT by calling subroutine CFFT96.asm N times, if an N by N 2D FFT is to be performed. Also, the code demonstrates the implementation of a double buffer by the DMA controllers on the DSP96002 as discussed in **SECTION C.5**.

C.7.2 Discrete Cosine Transform on the DSP96002

C.7.2.1 One Dimensional Discrete Cosine Transform (DCT) The one dimensional cosine transform of a discrete time sequence $x(n)$, $n = 0, 1, \dots, N-1$ is defined as:

$$F(k) = \frac{2c(k)}{N} \sum_{n=0}^{N-1} x(n) \cos \left[\frac{(2n+1)k\pi}{2N} \right]; \quad k = 0, 1, \dots, N-1 \quad (C.50)$$

where:

$$\begin{aligned} c(k) &= \frac{1}{\sqrt{2}}, & k &= 0 \\ &= 1, & k &= 1, 2, \dots, N-1 \\ &= 0, & & \text{elsewhere} \end{aligned}$$

and the inverse transform is:

$$x(n) = \sum_{k=0}^{N-1} c(k)F(k) \cos \left[\frac{(2n+1)k\pi}{2N} \right]; \quad n = 0, 1, \dots, N-1 \quad (\text{C.51})$$

A fast discrete cosine transform (FDCT) proposed by Chen and Smith [see Reference 1] is adapted in this application note, and its flow diagram is plotted in Figure C.35. Many optimized implementations on the FDCT have been published. The code given on the Motorola DSP bulletin board is not fully optimized; it simply demonstrates the simplicity of the DSP96002 assembly code.

For $N=2^m$, $m > 2$, this algorithm requires:

$(3N/2)(\log_2 N - 1) + 2$ real additions and
 $N \log_2 N - (3N/2) + 4$ real multiplications.

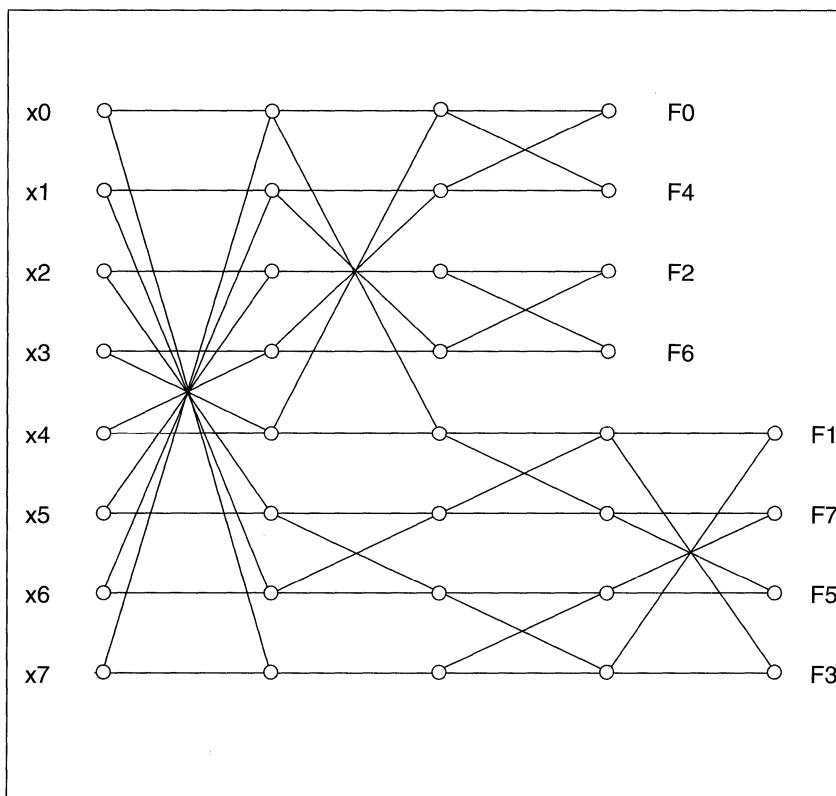


Fig. C.35 The flow diagram of an 8-point discrete cosine transform. Note that the output order of the transform is scrambled.

C.7.2.2 Two Dimensional DCT A one dimensional DCT can be easily extended to a two dimensional DCT as shown in Eqn. (C.52).

$$F(j, k) = \frac{4c(j)c(k)}{N^2} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(m, n) \cos \left[\frac{(2n+1)k\pi}{2N} \right] \times \cos \left[\frac{(2m+1)j\pi}{2N} \right] \quad (\text{C.52})$$

Therefore, to calculate an N by N 2D DCT, repeat the N-point 1D DCT N times. An 8x8 2D DCT assembly code for the DSP96002 (DCT.asm) is presented on the Motorola DSP bulletin board.

C.8 COMPETITIVE ANALYSIS OF FFT PERFORMANCES

"The total Icycles of the DSP56156 can be reduced to about 44000 Icycles and the twiddle factors can be cut to N/2 with further optimization."

C.8.1 Most Popular Digital Signal Processors

Currently a variety of DSPs are available from a dozen semiconductor vendors. This section addresses floating-point DSPs first, because the FFT is one of their most important benchmarks. The architecture of floating-point DSPs is optimized for FFTs.

Fixed-point DSPs are also discussed because they have a higher performance-to-cost ratio than the floating-point DSPs, and are used more frequently in DSP applications such as digital audio, speech processing, telecommunication, automobile control, and home electronics. Since cost is a very sensitive issue in fixed-point DSPs, some useful features such as address mode, instruction type, number of input operands in each operation, and I/O capability can be offset with a reduction in silicon area to keep the cost as low as possible.

In general, the FFT performance on fixed-point DSPs is less than floating-point DSPs if the comparison is conducted on DSPs from the same vendor. But it is not surprising that a fixed-point DSP from one manufacturer may offer a higher performance than a floating-point DSP from a different manufacturer. After comparing existing DSPs, one may decide which is an optimal architecture for FFTs regarding speed and cost, where cost refers to required memory speed, memory size, and silicon area for special hardware that aids FFT calculation. It is impractical to base the decision on selling prices because they can be strongly influenced by sales strategies of different DSP vendors.

The following sections compare DSPs from Motorola, Texas Instruments, AT&T, and Analog Devices. There are other DSPs from new players that may have their merits, but they are not included in the following discussion due to their short time on the market.

C.8.2 Performance of FFTs on Digital Signal Processors

Digital signal processors can be divided into two categories; floating-point DSPs and fixed-point DSPs. As is well known, the fixed-point DSPs suffer saturation prob-

lems in calculations. To solve this problem, the programmer must scale down input data either at the front or in the middle of the calculation, which results in a shrunken signal-to-noise ratio or dynamic range. The floating-point DSPs use an extra data section to hold exponent information, consequently, the dynamic range is so large that the chance of overflow is non-existent in most circumstances. Of course, one has to pay for this convenience by requiring wider data memory, a larger silicon area, and more power consumption.

C.8.2.1 FFTs on Floating-Point DSPs Steps to implement various floating-point DSPs may differ depending on their conformance with the IEEE 754-1985 standard. In general, an IEEE floating-point DSP requires more computational steps to generate a normalized result than a proprietary implementation does. Although the IEEE implementation may result in a bigger die design in achieving the same clock rate, it does, however, provide a standard interface to other microprocessors. In contrast, when proprietary formatted DSPs interface to other general purpose microprocessors, they require extra time to convert to the IEEE format. Motorola and Analog Devices are committed to the IEEE floating-point format. TI and AT&T use their own proprietary format.

Table C.1 offers a fair comparison of complex FFTs on the different floating-point DSPs. Note that there are no constraints on the FFT algorithm. The FFT can be a

Table C.1 1024-Point Complex FFT on Floating-Point DSPs

DSPs	96002 ¹	AD21020 ²	TIC40 ³	TIC30 ¹	AT&T32C ¹
Icycle (ns)	50	50	50	60	80
Algorithm	DIT	DIT	DIT	DIT	DIT
Radix	2	4	2	2	2
P Memory (word)	219	192	215	231	158
Data Memory (word)	4N	4N	4N	4N	2N
SIN/COS. table	3N/2	3N/2	N/2	N/2	N/2
Instruction length (bit)	32	48	32	32	32
SRAM for Zero Wait State (ns)	20	35	25	35	20
Total Icycles	18891	19245	30903	40457	35330
Total Time (ms)	0.94455	0.96225	1.54516	2.4274	2.8264

1. R.Meyer and K. Schwartz, "FFT implementation on DSP-Chips-Theory and Practice," ICASSP, 1990.

2. Analog Devices, ADSP-21020 User's Manual.

3. Texas Instruments, TMS320C4x User's Guide.

NOTE: Icycle in Table C.1 refers to instruction **cycle**.

Minimum Icycle denotes the reciprocosity of the highest clock frequency available on the DSPs.

Table C.2 1024-Point Real Input FFT on Floating-Point DSPs

DSPs	96002 ¹	TIC40 ²	TIC30 ³	AT&T32C ⁴
Icycle (ns)	50	50	60	80
Total Icycles	11600	20396	31317	26300
Total Time (ms)	0.58	1.01984	1.879	2.106

1. See RFFT96T.asm on the Motorola DSP Bulletin Board (Dr. BuB).

2. Texas Instruments, TMS4x User's Guide

3. Texas Instruments, Digital Signal Processing Applications with the TMS320 Family.

4. AT&T DSP32C User's Manual.

Decimation in Time (DIT) or Decimation in Frequency (DIF), and can also be a radix two or radix four butterfly, as long as the algorithm can generate the best performance on a specified processor.

Table C.1 shows that the Motorola DSP96002 performs the fastest 1024-point complex FFT. The Analog Devices' ADSP21020 performs almost as well as the DSP96002. The main factor that makes these two DSPs so fast in calculating the FFT is the special instruction "MPY| |ADD| |SUB." Supported by this instruction, the DSP96002 needs only four instruction cycles to perform one radix 2 butterfly, and the DSP21020 needs only fourteen instruction cycles to do one radix 4 butterfly. However, the DSP96002 has 2x512 data words on the chip and it features two on-chip DMA controllers. The on-chip memory and DMA controllers are extremely important features in implementing real-time data acquisition and control. The lack of peripherals and memory on the DSP21020 forces it into the position of competing with RISC chips. Although the DSP21020 requires lower cost SRAM for zero wait states interface, the program memory has to be 48-bits wide which negates the system cost benefits of using slow memory.

C.8.2.2 FFT on Fixed-Point DSPs As mentioned previously, scaling must be performed on the fixed-point DSPs to prevent overflow in the intermediate stage of calculation. The following benchmarks, either complex or real FFT, assume that each input data has been divided by the number of the FFT.

As shown in Table C.3, the Motorola DSP56001/2 has a minimum icycle time and uses only N/2 locations for both real and imaginary twiddle factors. The total Icycles of the DSP56156 can be reduced to about 44000 Icycles and the twiddle factors can be cut to N/2 with further optimization.

Table C.3 1024-Point Complex FFT on Fixed-Point DSPs

DSPs	56001/2 ¹	AD2100A ²	TIC25 ³	TIC50 ³	56156 ⁴
Icycle (ns)	60/50	80	80	35	33
Algorithm	DIT	DIT	DIT	DIT	DIT
Radix	2	4	2	2	2
P Memory (word)	234	222			158
Data Memory (word)	4N	4N	2N	2N	4N
SIN/COS table	N/2	3N/2	5N/4	5N/4	N
Instruction length (bit)	24	24	16	16	16
Total Icycles	29949	34625	113487	82761	46373
Total Time (ms)	1.79694/1.49745	2.77	9.079	2.8967	1.53031

1. See CFFT56.asm on the Motorola DSP Bulletin Board (Dr. BuB).

2. R. Meyer and K. Schwartz, "FFT Implementation on DSP Chips — Theory and Practice," ICASSP, 1990.

3. Texas Instruments TMS320 DSP Family Benchmarks.

4. See CFFT156.asm on the Motorola DSP Bulletin Board (Dr. BuB).

Table C.4 1024-Point Real Input FFT on Fixed-Point DSPs

DSPs	56002 ¹	TIC25 ²	TIC50 ²
Icycle (ns)	50	80	35
Total Icycles	17443	56286	48055
Total Time (ms)	0.87215	4.50288	1.6819

1. See RFFT56T.asm on the Motorola DSP Bulletin Board (Dr. BuB).

2. Texas Instruments TMS320 DSP Family Benchmarks.

C.9 CONCLUSION

"Motorola's family of digital signal processors, combined with Motorola's data conversion parts, provide a complete, cost-efficient solution to frequency domain problems . . ."

Frequency domain applications are becoming more important as inexpensive hardware solutions become more readily available. Motorola's Family of DSP56001/2 and DSP96002 digital signal processors provide particularly effective solutions to frequency domain problems. A highly parallel architecture, combined with an instruction


```

;*****
;
;include 'r4tab1'
;include 'rmax'
;
;points    equ 1024          ; FFT-length, only 1024 possible
;passes    equ 10            ; ld(points)
;data      equ $800          ; input data, normal order
;odata     equ $C00          ; output data, normal order
;tab4      equ $1000         ; start of radix-4 twiddle factors
;                          ; 766 complex values)
;
;r4  tab1  tab4
;
;org      p:$100
;
;move     #$008A0000,x:$FFFFFFD ; zero wait states in BCRB
;move     #$008A0000,x:$FFFFFFE ; zero wait states in BCRA
;move     #$0000FFFF,x:$FFFFFFC ; X port A, Y and P port B in PSR
;                          ;Upper three moves won't count for the benchmark,
;                          ;only for initializing the simulator or the DSP.
;                          ;They show how to configure the device.
;
;  A T T E N T I O N   P L E A S E  !!!!!!!
;
; STEP THROUGH THE FIRST THREE LINES, THEN LOAD THE SIMULATOR NEW
; WITH RMAXS AND INPUT VECTORS, THEN GET A NEW RUN
;
;rmxpoints,passes,data,odata,tab4
;
;nop
;nop
;nop
;
;end
;*****
;
;          COMPLEX, RADIX-2,4 DIT FFT   :  RMAX.ASM
;          -----
;
;MACRO FOR A FAST LOOPED-CODE MIXED-RADIX DIT FFT COMPUTATION
;          IN DSP96002
;
;WRITTEN BY: KARL SCHWARZ, RAIMUND MEYER      10.11.89
;
;LEHRSTUHL FUER NACHRICHTENTECHNIK
;UNIVERSITAET ERLANGEN-NUERENBERG
;
;REVISION : THIS PROGRAM IS SPEEDED UP FROM RMIX1.ASM
;
;PLEASE LOOK IN THE START FILE RMAXS.ASM HOW TO CONFIGURE THE DEVICE
;
;FOR THIS PROGRAM THE FFTLENGTH IS 1024 POINTS
;SPECIAL FEATURES : RADIX-4 BUTTERFLY IN FIRST AND LAST TWO STAGES
;SIMPLE RADIX-4 BUTTERFLY IN 1. TO 6. STAGE IF NO TWIDDLES ARE USED
;TABLE IN USE : ONLY R4TAB1.ASM FOR RADIX-2 AND LAST RADIX-4 BUTTERFLY
;LOOK IN R4TAB1.M HOW TO BUILT A TABLE
;*****
;
;EXAMPLE FOR THE 1024 POINT COMPLEX FFT (WITH BITREVERSAL) :
;
;MEMORY SIZE :  PROGRAM   : 219 WORDS
;              DATA     : 4096 WORDS
;              TWIDDLE FACTORS : 1532 WORDS
;
;CYCLES PER BUTTERFLY :
;1. AND 2. STAGE:      2
;3. AND 4. STAGE:     3.5
;5. AND 6. STAGE:    3.875
;7. STAGE :           4
;8. STAGE :           4.25

```

```

;      9. AND 10. STAGE :      4.25
;      AVERAGE CYCLES/BUTTERFLY:      3.55
;      TOTAL BUTTERFLYCYCLES :      18176
;      INITIALIZATION OVERHEAD:      715 = 3.8 % OF TOTAL TIME
;      TOTAL NUMBER OF INSTRUCTION CYCLES : 18891
;      TOTAL TIME FOR A 1024 POINT FFT:      1.399 msAT 27 MHz

```

```

*****
;
;      USED RADIX-2 BUTTERFLY
;
;      AR + j AI -----O-----+-----O----- AR' + j AI'
;                               \      /
;                               /      \
;                               \      /
;                               /      \
;      BR + j BI --- ( COS - j SIN ) ---O-----O----- BR' + j BI'
;
;      TR = BR * COS + BI * SIN
;      TI = BR * SIN - BI * COS
;      AR' = AR + TR
;      AI' = AI - TI
;      BR' = AR - TR
;      BI' = AI + TI

```

```

*****
;
;      USED RADIX-4 BUTTERFLY
;
;      AR + j AI -----O-----O-----O-----O--- AR' + j AI'
;                               \      /      \      /
;      BR + j BI --- (W1) ---O---X---O---O---O--- BR' + j BI'
;                               /      \      /      \
;      CR + j CI --- (W2) ---O---X---O---O---O--- CR' + j CI'
;                               /      \      /      \
;      DR + j DI --- (W3) ---O---O---(-j)---O---O--- DR' + j DI'
;
;      MIXING OF RADIX-2 AND RADIX-4 BUTTERFLIES IS POSSIBLE WITHOUT TROUBLE !

```

```

; ---> about 10 % faster than ICASSP 89 Paper 40.D9.7 by Kloker and Lindsley

```

```

rmaxmacropoints,passes,data,odata,tab4

```

```

;points : FFT-length (power of 2)
;passes : log2(points)
;data : start address of input vector
;odata : start address of output vector due to bitreversal
;tab4 : start address of radix-4 twiddle factors from file r4tab.asm

```

```

pam5 equ passes-5
pg2 equ points/2
pg4 equ points/4
pg8 equ points/8
pg 16 equ points/16
pg 32 equ points/32
pg 64 equ points/64
pg 128 equ points/128
pg 4m1 equ points/4-1
pg 16m1 equ points/16-1
pg 64m1 equ points/64-1

```

```

*****
; ----- FIRST 2 STAGES AS RADIX-4 BUTTERFLY ----- *
; *****

```

```

        move    #-1,m0
        move    m0,m1
        move    m0,m2
        move    m0,m3
        move    m0,m4
        move    m0,m5
        move    m0,m6
        move    m0,m7
        move    #data,r0
        move    #(data+pg4),r1
        move    #(data+2*pg4),r2
        move    #(data+3*pg4),r3
        move    #2,n0
        move    n0,n6
        move    n0,n5
        move    n0,n7
        move    #pg4m1,n1
jsr _sr4

;*****
; ----- PARTS OF 3. AND 4. STAGE AS SPECIAL RADIX-4 BUTTERFLY ----- *
;*****

        move    #data,r0
        move    #(data+pg16),r1
        move    #(data+2*pg16),r2
        move    #(data+3*pg16),r3
        move    #pg 16m1,n1
jsr _sr4

;*****
; ----- PARTS OF 5. AND 6. STAGE AS SPECIAL RADIX-4 BUTTERFLY ----- *
;*****

        move    #data,r0
        move    #(data+pg64),r1
        move    #(data+2*pg64),r2
        move    #(data+3*pg64),r3
        move    #pg 64m1,n1
jsr _sr4

;*****
; ----- REST OF 3. STAGE AS RADIX-2 BUTTERFLY ----- *
;*****

        move    #3,n6                ; step for twiddle addressing in r4tab

        move    #(tab4+3),r6          ; address of sin cos table
        move    #(data+pg4),r0        ; input vector
        move    #(data+pg4+pg8),r1
        move    #3,n7                ; still 3 r2 groups to calculate
        move    #(pg 8-3),r7          ; pg8 r2 butterflies in a group
        move    #(pg 8+1),n0          ; step to next group
jsr _nr2

;*****
; ----- REST OF 4. STAGE AS RADIX-2 BUTTERFLY ----- *
;*****

        move    #(tab4+6),r6          ; address of sin cos table
        move    #(data+pg4),r0        ; input vector
        move    #(data+pg4+pg16),r1
        move    #6,n7                ; still 6 r2 groups to calculate
        move    #(pg16-3),r7          ; pg16 r2 butterflies in a group
        move    #(pg16+1),n0          ; step to next group
jsr _nr2

;*****
; ----- REST OF 5. STAGE AS RADIX-2 BUTTERFLY ----- *
;*****

```



```

move    #(tab4+3),r6      ; address of sin cos table
move    #(data+pg16),r0   ; input vector
move    #(data+pg16+pg32),r1
move    #15,n7            ; still 15 r2 groups to calculate
move    #(pg32+3),r7      ; pg32 r2 butterflies in a group
move    #(pg32+1),n0      ; step to next group
jsr     _nr2

```

```

;*****
; ----- REST OF 6. STAGE AS RADIX-2 BUTTERFLY ----- *
;*****

```

```

move    #(tab4+6),r6      ; address of sin cos table
move    #(data+pg16),r0   ; input vector
move    #(data+pg16+pg64),r1
move    #30,n7            ; still 30 r2 groups to calculate
move    #(pg64+3),r7      ; pg64 r2 butterflies in a group
move    #(pg64+1),n0      ; step to next group
jsr     _nr2

```

```

;*****
; ----- 7. STAGE AS RADIX-2 BUTTERFLY ----- *
;*****

```

```

move    #(tab4),r6        ; address of sin cos table
move    #(data),r0        ; input vector
move    #(data+pg128),r1
move    #64,n7            ; still 64 r2 groups to calculate
move    #(pg128+3),r7     ; pg128 r2 butterflies in a group
move    #(pg128+1),n0     ; step to next group
jsr     _nr2

```

```

;*****
; ----- 8. STAGE AS RADIX-2 BUTTERFLY ----- *
;*****

```

```

move    #5,n0
move    #(data+4),r1
move    #tab4,r6
move    #data,r0
move    r1,r5
mover   0,r4
move    n0,n1
move    n0,n4
move    n0,n5

```

```

move    x:(r6),d9.s      y:,d8.s
move    x:(r1)+,d0.s     y:,d1.s
move    x:(r0),d4.s      y:(r6),d2.s
move    y:(r1),d7.s
faddsub.s d4,d0    x:(r1)+,d6.s    y:(r6)+n6,d3.s

```

```

do      #pg8,_end3      ; loop of groups

```

```

fmpy    d9,d6,d0        fsub.s    d1,d2    d0.s,x:(r4)    y:(r0)+,d5.s
fmpy    d9,d7,d1        faddsub.s d5,d2    d4.s,x:(r5)    y:(r1),d7.s
fmpy    d8,d6,d2        fadd.s    d3,d0    x:(r0),d4.s    d2.s,y:(r5)+
fmpy    d8,d7,d3        faddsub.s d4,d0    x:(r1)+,d6.s    d5.s,y:(r4)+

fmpy    d9,d6,d0        fsub.s    d1,d2    d0.s,x:(r4)    y:(r0)+,d5.s
fmpy    d9,d7,d1        faddsub.s d5,d2    d4.s,x:(r5)    y:(r1),d7.s
fmpy    d8,d6,d2        fadd.s    d3,d0    x:(r0),d4.s    d2.s,y:(r5)+

fmpy    d8,d7,d3        faddsub.s d4,d0    x:(r1)+n1,d6.s    d5.s,y:(r4)+
fmpy    d9,d6,d0        fsub.s    d1,d2    d0.s,x:(r4)    y:(r0)+,d5.s
fmpy    d9,d7,d1        faddsub.s d5,d2    d4.s,x:(r5)    y:(r1),d7.s
fmpy    d8,d6,d2        fadd.s    d3,d0    x:(r0),d4.s    d2.s,y:(r5)+
move    x:(r6)+n6,d9.s    y:,d8.s
fmpy    d8,d7,d3        faddsub.s d4,d0    x:(r1)+,d6.s    d5.s,y:(r4)+

```

```

    fmpy      d9,d6,d0      fsub.s   d1,d2      d0.s,x:(r4)   y:(r0)+n0,d5.s
    fmpy      d9,d7,d1      faddsub.s d5,d2      d4.s,x:(r5)   y:(r1),d7.s
    fmpy      d8,d6,d2      fadd.s    d3,d0      x:(r0),d4.s    d2.s,y:(r5)+n5

    fmpy      d8,d7,d3      faddsub.s d4,d0      x:(r1)+,d6.s   d5.s,y:(r4)+n4
_end3

;*****
;----- LAST TWO STAGES AS RADIX-4 BUTTERFLY ----- *
;*****

    move      #$0,m2
    move      m2,m3
    movem     2,m5
    movem     2,m7
    move      #data,r0
    move      #(data+1),r4
    move      #(data+2),r1
    move      #(tab4+1),r6
    move      #2,n4
    move      #4,n0
    move      n0,n1
    move      #odata,r5
    move      #(odata+pg2),r2
    move      #(odata+pg4),r7
    move      #(odata+pg4*3),r3
    move      #pg8,n5
    move      n5,n2
    move      n5,n7
    move      n5,n3

    move      x:(r4)+n4,d3.s   y:,d5.s   ;d3=Br,d5=Bi
    move      x:(r4)+n4,d1.s   y:,d2.s   ;d1=Dr,d2=Di
    faddsub.s d1,d3           x:(r0),d7.s   ;d3=Br+Dr,d1=Br-Dr,d7=Ar
    faddsub.s d5,d2           x:(r1),d0.sd1.s, y:(r7) ;d5=Bi+di,d2=Bi-Di,d0=Cr,
                                ;temp store Br-Dr
    faddsub.s d7,d0           d3.s,d4.s     y:(r1)+n1,d1.s ;d0=Ar+Cr,d7=Ar-
                                ;Cr,d4=Br+Dr,d1=Ci
    faddsub.s d7,d5           x:(r4),d6.s   y:(r0)+n0,d3.s ;d7=Ar-Cr-(bi+Di)
    faddsub.s d0,d4           d7.s,x:(r3)   y:(r4)+n4,d7.s

    do #pg4m1,_er4
    faddsub.s d3,d1           x:(r6)+,d9.sy:,d8.s
    fmpy.s    d6,d9,d5        d5.s,x:(r7)
    fmpy      d7,d8,d3        faddsub.s    d1,d2d4.s,   x:(r5)d3.s,d4.s
    fmpy      d6,d8,d1        fadd.s        d5,d3d0.s,   x:(r2)+n2d1.s,y:
    fmpy.s    d7,d9,d5        x:(r6)+,d9.s   y:,d8.s
    fsub.s    d1,d5           x:(r4)+n4,d6.sy:,d7.s
    fmpy.s    d6,d9,d1        y:(r7),d0.s
    fmpy      d7,d8,d2        faddsub.s    d4,d0      d2.s,y:(r5)+n5
    fmpy      d6,d8,d0        fadd.s        d2,d1      x:(r1),d6.sd0.s,y:(r7)+n7
    fmpy      d7,d9,d2        faddsub.s    d1,d3      x:(r6)+,d9.sy:,d8.s
    fmpy      d6,d9,d0        fsub.s        d0,d2      y:(r1)+n1,d7.s
    fmpy      d7,d8,d3        faddsub.s    d5,d2      d3.s,d4.d4.s,y:(r3)+n3
    fmpy      d7,d9,d1        fadd.s        d3,d0      x:(r0),d7.sd1.s,y:(r7)
    fmpy      d6,d8,d3        faddsub.s    d7,d0
    faddsub.s d7,d5
    faddsub.s d0,d4d7.s,      x:(r3)         y:(r4),d7.s
    fsub.s    d3,d1          x:(r4)+n4,d6.sy:(r0)+n0,d3.s

_er4
    faddsub.s d3,d1d5.s,     x:(r7)
    faddsub.s d1,d2         y:(r7),d6.s
    move      d0.s,         x:(r2)d1.s,   y:
    faddsub.s d3,d6d4.s,     x:(r5)d2.s,   y:
    move      d6.s,         y:(r7)
    move      d3.s,         y:(r3)

;_ponrjmp_ponr      ; REMOVE THIS COMMAND AND APPEND YOUR OWN JOB
    nop
    nop

```

```

        jmp *
;*****
;----- END OF FFT -----*
;*****
; SUBROUTINES FOLLOWING
;*****
;----- SPECIAL RADIX-4 BUTTERFLY WITH SIMPLE TWIDDLES -----*
;*****

_sr4
        move     r0,r4
        move     r1,r5
        move     r3,r7
        move     r2,r6

        move     y:(r5)+,d1.s
        move     x:(r0)+,d0.s      y:(r7)+,d3.s
        faddsub.s d1,d3      x:(r2),d2.s
        faddsub.s d0,d2      y:(r4),d5.s
        faddsub.s d0,d1      x:(r1),d4.s      y:(r6)+,d7.s
        faddsub.s d5,d7      d1.s,x:(r2)+
        faddsub.s d7,d3      x:(r3),d6.s      y:(r5)-,d1.s
        faddsub.s d6,d4      d0.s,x:(r3)+      d3.s,y:(r4)+
        faddsub.s d2,d4      x:(r0)-,d0.s      d7.s,y:(r5)+n5
        faddsub.s d5,d6      d2.s,x:(r1)+      y:(r7)-,d3.s

do n1,_st2
        faddsub.s d1,d3      x:(r2),d2.s      d5.s,y:(r7)+n7
        faddsub.s d0,d2      d4.s,x:(r0)+n0      y:(r4),d5.s
        faddsub.s d0,d1      x:(r1),d4.s      y:(r6)-,d7.s
        faddsub.s d5,d7      d1.s,x:(r2)+      d6.s,y:(r6)+n6
        faddsub.s d7,d3      x:(r3),d6.s      y:(r5)-,d1.s
        faddsub.s d6,d4      d0.s,x:(r3)+      d3.s,y:(r4)+
        faddsub.s d2,d4      x:(r0)-,d0.s      d7.s,y:(r5)+n5
        faddsub.s d5,d6      d2.s,x:(r1)+      y:(r7)-,d3.s

_st2
        move     d4.s,x:(r0)      d5.s,y:(r7)
        move     d6.s,y:(r6)
        rts

;*****
;----- NORMAL RADIX-2 BUTTERFLY -----*
;*****

_nr2
        move     r0,r4
        move     r1,r5
        move     n0,n1
        move     n0,n4
        move     n0,n5
        move     x:(r6)+n6,d9.s      y:,d8.s
        move     y:(r1),d7.s

        fmpy.s    d8,d7,d3      x:(r1)+,d6.s
        fmpy.s    d9,d6,d0
        fmpy.s    d9,d7,d1      y:(r1),d7.s
        fmpy      d8,d6,d2      fadd.s    d3,d0      x:(r0),d4.s

        fmpy      d8,d7,d3      faddsub.s d4,d0      x:(r1)+,d6.s

do n7,_endgrp                ; loop of groups
do r7,_bfly                  ; butterflyloop

        fmpy d9,d6,d0 fsub.s    d1,d2      d0.s,x:(r4)      y:(r0)+,d5.s
        fmpy d9,d7,d1 faddsub.s d5,d2      d4.s,x:(r5)      y:(r1),d7.s
        fmpy d8,d6,d2 fadd.s    d3,d0      x:(r0),d4.s      d2.s,y:(r5)+

```

```

    fmpy d8,d7,d3 faddsub.s    d4,d0    x:(r1)+,d6.s    d5.s,y:(r4)+
_bfly
    fmpy d9,d6,d0 fsub.s      d1,d2    d0.s,x:(r4)      y:(r0)+,d5.s
    fmpy d9,d7,d1 faddsub.s    d5,d2    d4.s,x:(r5)      y:(r1),d7.s
    fmpy d8,d6,d2 fadd.s       d3,d0    x:(r0),d4.s      d2.s,y:(r5)+

    fmpy d8,d7,d3 faddsub.s    d4,d0    x:(r1)+n1,d6.s    d5.s,y:(r4)+
    fmpy d9,d6,d0 fsub.s      d1,d2    d0.s,x:(r4)      y:(r0)+,d5.s
    fmpy d9,d7,d1 faddsub.s    d5,d2    d4.s,x:(r5)      y:(r1),d7.s
    fmpy d8,d6,d2 fadd.s       d3,d0    x:(r0),d4.s      d2.s,y:(r5)+

    move                                x:(r6)+n6,d9.s    y:.,d8.s

    fmpy d8,d7,d3 faddsub.s    d4,d0    x:(r1)+,d6.s    d5.s,y:(r4)+
    fmpy d9,d6,d0 fsub.s      d1,d2    d0.s,x:(r4)      y:(r0)+n0,d5.s
    fmpy d9,d7,d1 faddsub.s    d5,d2    d4.s,x:(r5)      y:(r1),d7.s
    fmpy d8,d6,d2 fadd.s       d3,d0    x:(r0),d4.s      d2.s,y:(r5)+n5
    fmpy d8,d7,d3 faddsub.s    d4,d0    x:(r1)+,d6.s    d5.s,y:(r4)+n4
_endgrp
    rts

    endm

% MATLAB-File to generate the radix-4 twiddle factor table for the
% fast FFT-program RMIX1.ASM .
% By increasing the variable fftlength you can make tables for higher
% FFT-lengths than 1024.
%
% Karl Schwarz, Raimund Meyer      17.10.1989
% Lehrstuhl fuer Nachrichtentechnik
% Universitaet Erlangen-Nuernberg

    fftlength=1024      ;
    fg4=fftlength/4     ;
    fg4ml=fg4-1         ;
    x=0:fg4ml           ;
    a=bitrev(x)         ;
    a=a(2:fg4)          ;
    i=1                 ;
    c(i)=1              ;
    s(i)=0              ;
    i=i+1               ;
    kon=2*pi/fftlength  ;
    for k=1:fg4ml
        c(i)=cos(kon*a(k)) ;
        s(i)=sin(kon*a(k)) ;
        i=i+1             ;
        c(i)=cos(kon*a(k)*3) ;
        s(i)=sin(kon*a(k)*3) ;
        i=i+1             ;
        c(i)=cos(kon*a(k)*2) ;
        s(i)=sin(kon*a(k)*2) ;
        i=i+1             ;
    end
    c=c'                ;% real part of twiddle factor (cos)
    s=s'                ;% imaginary part of twiddle factor (sin)
end

Optimized Complex FFT for the DSP56001/2
    page    132.60
    opt     nomd,mex,loc,nocex,mu

    include 'sincosc'
    include 'bitrevtd56'
    include 'gen56'
    include 'cfft56'

; Latest revision - 14-Oct.-92

    reset    equ    0
    start    equ    $40

```

```

POINTS    equ      512
IDATA     equ      $0
COEF      equ      $800
ODATA     equ      $1000

sincosc   POINTS,COEF
gen56     POINTS,IDATA

opt       mex
org       p:reset
jmp       start

org       p:start
movep    #0,X:$FFFE          ;0 wait states
bitrevtd56 POINTS,COEF
CFFT56    IDATA,COEF,POINTS,ODATA
    nop
    nop
    jmp *
    end

;
; Sine-Cosine Table Generator for rfft56.asm and cfft56.asm
;
; Last Update 10/28/92
;
sincosc   macro    points,coef
sincosc   ident    1,2
;
;       sincosc -    macro to generate sine and cosine coefficient
;                   lookup tables for Decimation in Time complex FFT
;                   twiddle factors. Only points/4 coefficients
;                   are generated. For real FFT another points/4
;                   coefficients with higher freq. are created.
;
;       points -    number of points (2 - 32768, power of 2)
;       coef -      base address of sine/cosine table
;                   positive cosine value in X memory
;                   positive sine value in Y memory
;
;       8/12/92
;
    pi      equ      3.141592654
;freq      equ      2.0*pi/@cvf(points*2)

;       org       x:coef-points/2
;count     set      0
;       dup       points/2
;       dc        @cos(@cvf(count)*freq)
;count     set      count+1
;       endm

;       org       y:coef-points/2
;count     set      0
;       dup       points/2
;       dc        -@sin(@cvf(count)*freq)
;count     set      count+1
;       endm

freq1      equ      2.0*pi/@cvf(points)

; int i,j=1,k,tmp=0;
; k=1<<(length-1);
; for(i=0;i<length;i++){
;     if (integer&j) tmp=tmp|k;
;     j=j<<1;
;     k=k>>1;
;     org       x:coef
;count     set      0
;       dup       points/4
;       dc        @cos(@cvf(count)*freq1)
;count     set      count+1
;       endm

```

```

count    org    y:coef
count    set     0
count    dup     points/4
count    dc      @sin(@cvf(count)*freq1)
count    set     count+1
count    endm

;end of sincosr macro

bitrevtd56 macro POINTS,COEF
bitrevtd56 ident 1,2
;
;   bitrevtd - macro to sort sine and cosine coefficient
;               lookup tables in bit reverse order for 56156
;
;   POINTS -    number of points (2 - 32768, power of 2)
;   COEF -      base address of sine/cosine table
;               negative cosine (Wr) and negative sine (Wi) in X memory
;
; Wei Chen
; July-28, 1992
;

        move    #COEF,r1                ;twiddle factor start address
        move    #0,m0                  ;bit reverse address
        move    #POINTS/8,n0           ;sincosr use N/4 points data,
                                        ;offset for bit rev. is N/8

        move    #POINTS/4-1,n2
        move    r1,r0                  ;r1 ptr to normal order data
        move    (r1)+                  ;no swap on 1st data
        move    (r0)+n0                ;r0 ptr to bitrev
        do      n2,_end_bit            ;does N/4-1 points swap
        move    r1,x0
        move    r0,b
        cmp     x0,b
        jgt     _swap
        move    (r1)+                  ;no swap but update points
        move    (r0)+n0
        jmp     _nothing

_swap
        move    r1,r5
        move    r0,r4
        move    x:(r1),x0      y:(r5),y0
        move    x:(r0),a      y:(r4),b
        move    x0,x:(r0)+n0  y0,y:(r4)
        move    a,x:(r1)+      b,y:(r5)

_nothing
        nop
_end_bit
        endm                ;end of bitrevtd macro

; Input signal for FFT rfft56.asm and cfft56.asm
;
; Last Update 10/28/92
;
gen56 macro POINTS,IDATA
;
;   gen56 - macro to generate input signal for FFT test on 56001
;           2000 Hz sinewave with scaling factor POINTS in X and Y memory
;
;   POINTS - number of points (2 - 32768, power of 2)
;   IDATA -  base address of signal
;
srate set 44100 ;Hz
ffreq set 2000 ;Hz
ppi equ 3.141592654
freq2 equ 2.0*ppi*ffreq/@cvf(srate)

```

```

count    org    x:IDATA
        set     0
        dup     POINTS
        dc      @sin(@cvf(count)*freq2)/POINTS
count    set     count+1
        endm

count    org    y:IDATA
        set     0
        dup     POINTS
        dc      @sin(@cvf(count)*freq2)/POINTS
count    set     count+1
        endm

        endm    ;end of gen56 macro

```

```

;
; 512-Point, 28174 clock cycles Non-In-Place FFT.
;
; Sept. 11 92   Version 1.0
;
CFFT56 macro    IDATA,COEF,POINTS,ODATA
CFFT56 ident    1,0
;
; 512 Point Complex Fast Fourier Transform Routine
; using the Radix 2, Decimation in Time, Cooley-Tukey FFT algorithm.
;
; This routine performs a 512 point complex FFT by taking advantages of
; 1). internal memory access by starting first half data at location 0,
;    avoid cycle stretching;
; 2). using N/4 complex twiddle factors based on the fact that two
;    consecutive twiddle factors in DIT FFT has a difference -j
; 3). trivial twiddle factors (1,0) and (0,-1) are utilized.
;
;
; Complex input and output data
;   Real data in X memory
;   Imaginary data in Y memory
; Normally ordered input data
; Bit reversed output data for 1024 real input FFT
; Coefficient lookup table
;   +Cosine values in X memory
;   -Sine values in Y memory
;
;
; Address pointers are organized as follows:
;
;   r0 = ar,ai input pointer      n0 = group offset      m0 = modulo (points)
;   r1 = br,bi input pointer      n1 = group offset      m1 = modulo (points)
;   r2 = ext. data base address   n2 = groups per pass   m2 = 256 pt fft counter
;   r3 = coef. offset each pass  n3 = coefficient base addr. m3 = linear
;   r4 = ar',ai' output pointer   n4 = group offset      m4 = modulo (points)
;   r5 = br',bi' output pointer   n5 = group offset      m5 = modulo (points)
;   r6 = wr,wi input pointer      n6 = coef. offset      m6 = bit reversed
;   r7 = not used (*)             n7 = not used (*)       m7 = not used (*)
;
;   * - r7, n7 and m7 are typically reserved for a user stack pointer.
;
; Alters Data ALU Registers
;
;   x1    x0    y1    y0
;   a2    a1    a0    a
;   b2    b1    b0    b
;
; Alters Address Registers
;
;   r0    n0    m0
;   r1    n1    m1
;   r2    n2    m2
;   r3    n3    m3

```

```

;      r4      n4      m4
;      r5      n5      m5
;      r6      n6      m6
;
; Alters Program Control Registers
;      pc      sr
;
; Uses 8 locations on System Stack
;
;
;-----;
; Initialize pointers to r0->Ar,r1->Cr,r4->Bi,r5->Di, and r3->temp location ;
; r0,r1,r4, and r5 are modular addressing with modulo N/2 ;
;-----;

move #IDATA,r0      ;r0 -> Ar
move r0,n3
move #ODATA,r3      ;r3 always has ODATA
move #COEF+1,n6      ;n6 always has COEF,(0,1) is not used
move #POINTS/4,n0    ;offset and butterflies per group
move #POINTS/2-1,m0  ;modulo addressing
move r0,r6           ;r6=0 flag reg. for trivial groups

do #3,_end_trivial   ;do three R4 passes
move n0,n1           ;pointer offset
move n0,n4           ;pointer offset
move n0,n5           ;
lea (r0)+n0,r4        ;r4 -> Bi
move m0,m5
lea (r4)+n4,r1        ;r1 -> Cr
move m0,m1           ;
move m0,m4
lea (r1)+n1,r5        ;r5 -> Di
;-----;

; First two passes are combined into a R4 pass without multiplication
; because Wr=1,Wi=0 in first R2 pass and Wr=0, Wi=-1 in 2nd R2 pass
;
; Ar'=Ar+Cr+(Br+Dr) Br'=Ar+Cr-(Br+Dr)Cr'=(Ar-Cr)+(Bi-Di) Dr'=(Ar-Cr)-(Bi-Di)
; Ai'=Ai+Ci+(Bi+Di) Bi'=Ai+Ci-(Bi+Di)Ci'=(Ai-Ci)+(Dr-Br) Di'=(Ai-Ci)-(Dr-Br)
;
; This two passes fully utilize internal memory by storing input data at location 0
; For 1024-point complex FFTs, only 256-point in internal, rest of them in
; external, 17+2 instructions are needed for one butterfly because first and next to
; the last instruction in the loop takes two Icycles. Other parallel move seems to
; take two cycles, but one of the two moves is internal, only one cycle is needed.
; 4.75 Icycles per R2 butterfly in the first two passes.
;
; For 512-point complex FFT, 17+1 instructions are used because first instruction in
; the loop takes only one Icycle. 4.5 Icycles per R2 butterfly.
;
; For 256 or less point complex FFT, 17 Icycles are needed. 4.25 Icycles/bfly.
;-----;

move x:(r0)+n0,a      ; a= Ar r0 -> Br
move x:(r1)+n1,b      ; b= Cr,r1 -> Dr

do n0,_twopass
add a,b x:(r0)+n0,x1 y:(r5)+n5,y1 ;b=Ar+Cr,x1=Br,y1=Di,r0->Ar,r5->Ci
subl b,a b,x:(r0) y:(r4),b ;a=Ar-Cr,save Ar+Cr temp in Ar,b=Bi
add y1,b a,x0 y:(r4)+n4,a ;b=Bi+Di,x0=Ar-Cr,a=Bi again,
;save Ar-Cr in Dr,r4->Ai
sub y1,a b,x:(r3)x0,b ;a=Bi-Di,store Bi+Di temp in x:ODATA,b=Ar-Cr
sub a,b x:(r1),x0 ;b=Ar-Cr-(Bi-Di)=Dr',x0=Dr,r0 -> Ar
addl b,a b,x:(r1)+n1 x0,b ;a=Ar-Cr+(Bi-Di)=Cr',save Dr',b=Dr,r1->Cr
sub x1,b a,x:(r1)+ x0,a ;b=Dr-Br,save Cr',a=Dr,r1->nCr
add x1,a x:(r0)+n0,b b,y1 ;a=Dr+Br,b=Ar+Cr, y1=Dr-Br,r0->Br
sub a,b y:(r5),y0 ;a=Ar+Cr-(Dr+Br)=Br',y0=Ci
addl b,a b,x:(r0)+n0 y:(r4),b ;a=Ar+Cr+(Dr+Br)=Ar',save Br',r0->Ar,b=Ai
sub y0,b a,x:(r0)+ y:(r4),a ;b=Ai-Ci,a=Ai again, save Ar',r0->Ar
add y0,a x:(r3),b b,y0 ;a=Ai+Ci,y0=Ai-Ci,b=Bi+Di, r5->Ci
add a,b ;b=Ai+Ci+(Bi+Di)=Ai'

```



```

subl b,a y0,b      b,y:(r4)+n4      ;a=Ai+Ci-(Bi+Di)=Bi',b=Ai-Ci,save Ai'
add y1,b y0,a      a,y:(r4)+      ;b=Ai-Ci+(Dr-Br)=Ci',a=Ai-Ci,save Bi',r4->nBi
sub y1,a x:(r1)+n1,b b,y:(r5)+n5;a=Ai-Ci-(Dr-Br)=Di',b=nCr, save Ci'
move x:(r0)+n0,a    a,y:(r5)+;a=nAr, save Di',r5->nDi
_twopass
;-----;
; Do rest of trivial group by 5 1cyc butterfly
;-----;
move n5,a          ;n5 contains ptr to Ar already
asr a n5,r1        ;r1->Ar
move a,n1          ;get offset
move r1,r5         ;r5->Ai
lea (r1)+n1,r4     ;r4->Bi
move #2,n4         ;for pointer
move r4,r0         ;r0->Br
move x:(r1),a y:(r4)+b ;a=Ar,b=Bi,r4->nBi
do n1,_no_more     ;w=(0,-1), R2 butterfly
add a,b x:(r0),x0 y:(r4)-,y0 ;b=Ar+Bi=Ar',x0=Br,y0=nBi
subl b,a b,x:(r1)+ y:(r5),b ;a=Ar-Bi=Br',save Ar',b=Ai
add x0,b a,x:(r0)+ y:(r5),a ;b=Ai+Br=Bi',save Br',a=Ai again
subl b,a y0,b      b,y:(r4)+n4      ;a=Ai-Br=Ai',save Bi',b=nBi
move x:(r1),a a,y:(r5)+ ;a=nAr,save Ai'
_no_more
move n0,a
asr a n3,r0        ;r0->IDATA
asr a a,r2
move a,n0          ;(points in a group)/4 after a radix 4 pass
move x:(r2)-,b     ;dec r2
move r2,m0
_end_trivial
move r0,r4         ;output pointer
move n1,r1         ;r1->Br
move r1,r5         ;r5->Bi
move x:(r0),a      ;a=Ar
move x:(r1),b      ;b=Br
do n1,_extra      ;w=(1,0)
add a,b           ;b=Ar+Br=Ar', y0=Bi
subl b,a b,x:(r0)+ y:(r4),b ;a=Ar-Br=Br',save Ar',b=Ai
add y0,b a,x:(r1)+ y:(r4),a ;b=Ai+Bi=Ai',save Br',a=Ai
sub y0,a x:(r1),b b,y:(r4)+ ;a=Ai-Bi=Bi',save Ai',b=nBi
move x:(r0),a a,y:(r5)+ ;a=nAr,save Bi'
_extra
;-----;
; Remaining passes are broken down to POINTS/256 sets,
; each set has 256-point R2 FFT
; and runs on internal data and external coefficients.
; In each pass, first two groups take advantages of trivial twiddle factors and ;
; no multiplication is carried out. Remaining groups use complex twiddle factors.;
;-----;
; Radix 2, Decimation In Time Cooley-Tukey FFT algorithm
;-----;
; Ar,Ai ----> Radix-2 ----> Ar'=Ar + Wr*Br - Wi*Bi
; Br,Bi ----> Butterfly ----> Ai'=Ai + Wi*Br + Wr*Bi
;                               Br'=Ar - Wr*Br + Wi*Bi = 2*Ar - Ar'
;                               Bi'=Ai - Wi*Br - Wr*Bi = 2*Ai - Ai'
;                               ^
;                               |
;                               W=Wr-jWi
;-----;
; r0->A,r1->B,r4->A',r5->B',r6->TF,n0=offset for B pointer,
; n2=number of bflies in a group
; n3=number of groups in a pass, m3=number of pass, r2=n3 or n3+1
;-----;
move m2,m0          ;linear address
move m2,m1
move m2,m4
move m2,m5
move #POINTS/4,r0   ;start location of a pass
move #4,m3          ;4 passes in first 256-point
move #POINTS/16,n0  ;offset to point to Br and Bi

```

```

move    n0,n1
move    n0,n4
move    n0,n5
move    n6,r6           ;r6->COEF
move    n0,n2           ;number of bflies in the first pass=R2 bflies/4
lea     (r0)+n0,r1       ;r1->Br
move    r0,r4           ;r4->Ai
lea     (r1)-,r5         ;r5->Bi-1 for pointer reason
jsr     _body

;-----;
; The second 256-point FFT has no any trivial twiddle factors,
; three nested loops do it
;-----;

move    #256,r0          ;start location of first pass in 2nd 256
move    #5,m3            ;5 passes in second 256-point
move    #POINTS/8,n0      ;offset to point to Br and Bi
move    #COEF+1,r6        ;twiddle factor pointer
move    n0,n1
move    n0,n4
move    #IDATA,r4         ;r4->A' =IDATA
move    n0,n5
lea     (r4)+n4,r5        ;r5->B'
lea     (r0)+n0,r1        ;r1->B
move    x:(r5)-,a         ;r5->B'-1 for pointer reason
jsr     _body
jmp     _end_FFT

;-----;
; All subroutines
;-----;

_body
move    #1,n3            ;number of groups in a pass
move    n3,r2            ;copy of n3
jset    #0,m3,_set_grp;first 256-point has number of group 1,3,7,15,...
move    #2,r2

_set_grp
do      m3,_inner_loop
jsr     _inner_pass
move    n0,a
asr     a                #IDATA,r0;r0=IDATA
move    a,n0             ;n0=offset of B
move    r2,a
asl     a                n0,n1
move    a,r2             ;r2=r2*2
asr     b                r2,n3 ;n3=number of groups in second 256
jset    #0,m3,_inner_set ;set up start address,for 2nd 256-point r0 is already ok
lea     (r2)-,n3          ;n3=number of groups in first 256
move    n4,a
asl     a                n6,r6 ;for 1st 256, TF always starts at first location
move    a,r0             ;r0=start location of first 256-point

_inner_set
move    n0,n4
move    n0,n5
move    n0,r4
lea     (r0)+n0,r1        ;r1->B
move    r0,r4             ;r4->A'
lea     (r1)-,r5          ;r5->B'

_inner_loop

;-----;
; End inner loop
;-----;

move    #IDATA,r0
move    #32,n2            ;n2=number of groups in the next to last
                        ;pass for 1st 256
jset    #0,m3,_no_set     ;set up start address of TF,for 2nd
                        ;256-point r6 is already ok
move    #COEF,n6          ;now n6 -> COEF
move    n6,r6             ;r6=COEF
lea     (r0)+n0,r1        ;r1->Br
move    r0,r4             ;r4->Ai

```

```

_no_set
move #-1,r5                      ;r5->Bi
move #3,n0
move n0,n1
move n0,n4
move n0,n5
jsr _next_last                    ;do the pass next to last
;-----;
; End_next_last pass
;-----;
move #IDATA,r0                    ;r0->IDATA
move r3,r4                        ;r4->A',output ptr -> external memory
jclr #0,m3,_add_offset            ;set up output address for 2nd 256-point
move #256,n3
move r6,n6                        ;start address of TF for 2nd 256
lea (r3)+n3,r4
_add_offset
lea (r0)+,r1                      ;r1->B
lea (r4)-,r5                      ;r5->B'
move #64,n2                      ;number of blies in the last pass
move #2,n0
move n0,n1
move n0,n4
move n0,n5
move n6,r6                        ;r6=COEF
jsr _last
rts

_inner_pass
do n3,_end_grp                    ;do groups in a pass
move x:(r5),a                    ;for pointer reason,a=something
move x:(r6),x0 y:(r0),b          ;x0=Wr,b=Ai
move x:(r1),x1 y:(r6)+,y0        ;x1=Br,y0=Wi
do n0,_end_bfy1                  ;Radix 2 DIT butterfly kernel
;with y0=Wi,x0=Wr
;b=Ai-BrWi,y1=Bi, r1->nBi
;b=Ai-BrWi+BiWr=Ai',
;save prev.Br',a=Ai
;a=2Ai-Ai'=Bi',b=Ar,save Ai'
;b=Ar+BrWr,a=Ar,save Bi',r0->nAi
;b=Ar+BrWr+BiWi=Ar',x1=nBr
;a=2Ar-Ar'=Br',
;save Ar',b=nAi,r4->nAr

mac -x1,y0,b x:(r1)+,y1          ;save preve. Br' inc r5 and r1
macr x0,y1,b a,x:(r5)+ y:(r0),a ;inc r0,r4
;xl=nGBr
;for pointer reason,
;a=something,b=nGAR
;W=-j*W
;b=Ai-BrWr,y1=Bi,r1->nBi
;b=Ai-BrWr-BiWi=Ai',
;save prev. Br',a=Ai
;a=2Ai-Ai'=Bi',b=Ar,save Ai'
;b=Ar-BrWi,a=Ar,save Bi',r0->nAi
;b=Ar-BrWi+BiWr=Ar',x1=nBr
;a=2Ar-Ar'=Br',
;save Ar',b=nAi,r4->nAr

_end_bfy1
move a,x:(r5)+n5 y:(r1)+n1,b    ;save preve. Br' inc r5 and r1
move x:(r4)+n4,a y:(r0)+n0,b    ;inc r0,r4
move x:(r1),x1
move x:(r5),a y:(r0),b
do n0,_end_bfy2
mac -x1,x0,b y:(r1)+,y1          ;b=Ai-BrWr,y1=Bi,r1->nBi
macr -y0,y1,b a,x:(r5)+y:(r0),a ;b=Ai-BrWr-BiWi=Ai',
;save prev. Br',a=Ai
;a=2Ai-Ai'=Bi',b=Ar,save Ai'
;b=Ar-BrWi,a=Ar,save Bi',r0->nAi
;b=Ar-BrWi+BiWr=Ar',x1=nBr
;a=2Ar-Ar'=Br',
;save Ar',b=nAi,r4->nAr

_end_bfy2
move a,x:(r5)+n5 y:(r1)+n1,b    ;save preve. Br' inc r5 and r1
move x:(r4)+n4,a y:(r0)+n0,b    ;inc r0,r4
_end_grp
rts

_next_last
move x:(r5),a y:(r0),b          ;a=something,b=Ai
move x:(r1),x1 y:(r6),y0        ;x1=Br,y0=Wi
do n2,_n_last                    ;do the pass next to last,
;internal to internal
mac -x1,y0,b x:(r6)+,x0 y:(r1)+,y ;b=Ai-BrWi,x0=Wr,y1=Bi, r1->nBi
macr x0,y1,b a,x:(r5)+n5 y:(r0),a ;b=Ai-BrWi+BiWr=Ai',
;save prev. Br',a=Ai

```

```

subl b,a      x:(r0),b      b,y:(r4)      ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
macr x1,x0,b  x:(r0)+,a      a,y:(r5)      ;b=Ar+BrWr,a=Ar,save Bi',r0->nAi
macr y1,y0,b  x:(r1),x1      ;b=Ar+BrWr+BiWi=Ar',x1=nBr
subl b,a      b,x:(r4)+      y:(r0),b      ;a=2Ar-Ar'=Br',
;save Ar',b=nAi,r4->nAr

macr -x1,y0,b y:(r1)+n1,y1 ;b=Ai-BrWi,y1=Bi, r1->nGBi
macr x0,y1,b  a,x:(r5)+      y:(r0),a      ;b=Ai-BrWi+BiWr=Ai',
;save prev. Br',a=Ai
subl b,a      x:(r0),b      b,y:(r4)      ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
macr x1,x0,b  x:(r0)+n0,a      a,y:(r5)      ;b=Ar+BrWr,a=Ar,
;save Bi',r0->nGAi
macr y1,y0,b  x:(r1),x1      ;b=Ar+BrWr+BiWi=Ar',x1=nGBr
subl b,a      b,x:(r4)+n4      y:(r0),b      ;a=2Ar-Ar'=Br',save Ar',
;b=nGAi,r4->nGAR

macr -x1,x0,b y:(r1)+,y1      ;b=Ai-BrWr,y1=Bi,r1->nGBi
macr -y0,y1,b a,x:(r5)+n5y:(r0),a ;b=Ai-BrWr-BiWi=Ai',
;save prev. Br',a=Ai,r5->nGBi
subl b,a      x:(r0),b      b,y:(r4)      ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
macr -x1,y0,b x:(r0)+,      a,y:(r5)      ;b=Ar-BrWi,a=Ar,save Bi',r0->nGAi
macr y1,x0,b  x:(r1),x1      ;b=Ar-BrWi+BiWr=Ar',x1=nBr
subl b,a      b,x:(r4)+      y:(r0),b      ;a=2Ar-Ar'=Br',
;save Ar',b=nAi,r4->nGAR

macr -x1,x0,b y:(r1)+n1,y1 ;b=Ai-BrWr,y1=Bi,r1->nGBi
macr -y0,y1,b a,x:(r5)+      y:(r0),a      ;b=Ai-BrWi-BiWr=Ai',
;save prev. Br',a=Ai,r5->Bi
subl b,a      x:(r0),b      b,y:(r4)      ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
macr -x1,y0,b x:(r0)+n0,a      a,y:(r5)      ;b=Ar-BrWi,a=Ar,
;save Bi',r0->nGAi
macr y1,x0,b  x:(r1),x1      y:(r6),y0      ;b=Ar-BrWi+BiWr=Ar',
;x1=nBr,y0=nWi
subl b,a      b,x:(r4)+n4      y:(r0),b      ;a=2Ar-Ar'=Br',save Ar',
;b=nAi,r4->nGAR

_n_last
move          a,x:(r5)
rts

_last
move x:(r5),a y:(r0),b      ;a=something,b=Ai
move x:(r1),x1,y:(r6),y0    ;x1=Br,y0=Wi
do n2,_end_last             ;do last pass, internal to external
macr -x1,y0,b x:(r6)+,x0      y:(r1)+n1,y1 ;b=Ai-BrWi,x0=Wr,y1=Bi, r1->nGBi
macr x0,y1,b  a,x:(r5)+n5      y:(r0),a      ;b=Ai-BrWi+BiWr=Ai',
;save prev. Br',a=Ai
subl b,a      x:(r0),b      b,y:(r4)      ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
macr x1,x0,b  x:(r0)+n0,a      a,y:(r5)      ;b=Ar+BrWr,a=Ar,save Bi',r0->nGAi
macr y1,y0,b  x:(r1),x1      ;b=Ar+BrWr+BiWi=Ar',x1=nGBr
subl b,a      b,x:(r4)+n4      y:(r0),b      ;a=2Ar-Ar'=Br',
;save Ar',b=nGAi,r4->nGAR

macr -x1,x0,b y:(r1)+n1,y1 ;b=Ai-BrWr,y1=Bi,r1->nGBi
macr -y0,y1,b a,x:(r5)+n5      y:(r0),a      ;b=Ai-BrWr-BiWi=Ai',
;save prev. Br',a=Ai,r5->Bi
subl b,a      x:(r0),b      b,y:(r4)      ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
macr -x1,y0,b x:(r0)+n0,a      a,y:(r5)      ;b=Ar-BrWi,a=Ar,save Bi',r0->nGAi
macr y1,x0,b  x:(r1),x1      y:(r6),y0      ;b=Ar-BrWi+BiWr=Ar',
;x1=nBr,y0=nWi
subl b,a      b,x:(r4)+n4      y:(r0),b      ;a=2Ar-Ar'=Br',
;save Ar',b=nAi,r4->nGAR

_end_last
move          a,x:(r5)
rts
_end_FFT
endm

```

C.11 REAL-VALUED INPUT FFT

C.11.1 Faster Real FFT for the DSP96002

Fig. C.37 Faster real FFT for the DSP96002

```

page 132,60,1,1
opt mex
;*****
;Motorola Austin DSP Operation  20 August 1992
;*****
;Test program for DSP96002 rfft96.asm
;*****
;      1024 real-valued inputs
;      Maximum sample rate:  0.58 ms at 40.0 MHz
;      Memory Size:   Prog:141 + 32 words ;
;                      Data:2*1024 words(idata+odata) + 256 words (twiddle factor)
;      Number of clock cycles: 23200 (11600 instruction cycles)
;      Clock Frequency: 40.0MHz
;      Instruction cycle time: 50.ns
;*****
;
;      Real-Valued Input Radix 2 Cooley-Tukey Decimation in Time FFT
;
;
;      normally ordered input data
;      normally ordered output data
;
;*****
; Equates Section
;*****
RESET      equ      $00000000      ; reset isr
MAIN       equ      $00000100      ; main routine

points     equ      512            ;points=real data number /2
passes     equ      9              ;log2(points)=passes
idata      equ      $0
odata      equ      $1000
coef       equ      $800

BCRA       equ      $FFFFFFFE      ; port a bus control reg
BCRB       equ      $FFFFFFFD      ; port b bus control reg
PSR        equ      $FFFFFFFC      ; port select reg

include    'sincosf.asm'           ;using external cos and sin table,
                                   ;if use internal ROM, delete this line

include    'gen96.asm'
include    'cfft96.asm'
include    'split96.asm'

;*****
;*****
sincosf    points,coef      ;twiddle factor for split is a full cycle sin and cos
gen96      points,idata

org p:MAIN
movep #$0,x:BCRA           ; no wait states for portb P,X,Y,I/O
movep #$0,x:BCRB           ; ...don't care about page fault
movep #$0FF00FF,x:PSR      ; external X:memory on Port-B
                                   ;      Y:memory on Port-A
bclr #$3,omr               ; disable the internal data ROMs

CFFT96    points,passes,idata,coef,odata
SPLIT96   points,coef,odata

```

```

        nop
        nop
        jmp      *

        END

;
; Sine-Cosine Table Generator for rfft96.asm
;
; Last Update 5 August 92
;
sincosf macro    points,coef
sincosf ident    1,2
;
;      sincosf-    macro to generate sine and cosine coefficient
;                  lookup tables for Decimation in Time FFT
;                  twiddle factors.
;
;      points -    number of points (2 - 32768, power of 2)
;      coef  -    base address of sine/cosine table
;                  positive cosine value in X memory
;                  positive sine value in Y memory
;
;      8/12/92

pi      equ      3.141592654
freq    equ      2.0*pi/@cvf(points*2)

        org      x:coef-points/2
count   set      0
        dup      points/2
        dc       @cos(@cvf(count)*freq)
count   set      count+1
        endm

        org      y:coef-points/2
count   set      0
        dup      points/2
        dc       -@sin(@cvf(count)*freq)
count   set      count+1
        endm

freq1    equ      2.0*pi/@cvf(points)

        org      x:coef
count   set      0
        dup      points/2
        dc       -@cos(@cvf(count)*freq1)
count   set      count+1
        endm

        org      y:coef
count   set      0
        dup      points/2
        dc       -@sin(@cvf(count)*freq1)
count   set      count+1
        endm

        endm                                ;end of sincosf macro

;*****
;
; Split N/2 Complex FFT(Hn) for N real FFT(Fn)
;
SPLIT96 macro points,coef,odata
SPLIT96 ident 1,2
;
; Fi=0.5(Hi+Hn/2-i*)-0.5j(Hi-Hn/2-i*)W i=0,1,..,N-1
;
; points is real data /2
;

```

```

move #points-1,n0                ;number of complex FFT input data
move #points/2-1,n4              ;loop counter
move #odata,r0                  ;r0 ptr to A=Hi
move r0,r4                      ;r4 ptr to A'
move #-1,m6                    ;linear address
move m6,m5      move      m6,m4
move #coef-points/2+1,r6        ;twiddle factor start location
lea (r0)+n0,r1                 ;r1 ptr to B= Hn/2-1
move r1,r5                    ;r5 ptr to B'
move x:(r0)+,d0.s y:.,d1.s     ;DC=Ar0+Ai0
faddsub.s d0,d1      x:(r0)+,d2.s y:.,d3.s ;d0=Niquet=Ar0-Ai0,
move d1.s,x:(r4)+ d0.s,y:      ;d1=DC,d2=Ar,d3=Ai
                                ;save DC and Niq

move x:(r1)-,d7.s y:.,d1.s      ;d7=Br,d1=Bi
faddsub.s d7,d2      x:(r6)+,d8.s y:.,d9.s ;d7=Br-Ar=-H2i,
                                ;d2=Ar+Br=H1r,d8=Wr,d9=Wi<0
fmpy d9,d7,d0 faddsub.s d3,d1    d2.s,d5.s ;d0=Wi*H2i,d3=Ar-Bi=H1i,
                                ;d1=Ar+Bi=H2r,d5=H1r
fmpy.sd8,d1,d1 d1.s,d6.s        ;d1=Wr*H2r,d6=H2r
move #0.5,d4.s                 ;d4=0.5
;-----
; H1r=Ar+Br, H1i=Ar-Bi, H2r=Ar+Bi, H2i=Ar-Br
; Ar'=(H1r+Wr*H2r-Wi*H2i)/2
; Br'=(H1r-(Wr*H2r-Wi*H2i))/2
; Ai'=(Wi*H2r-Wr*H2i+H1i)/2
; Bi'=((Wi*H2r-Wr*H2i)-H1i)/2
;-----
do n4,_end_split
fmpy d8,d7,d2 fsub.s d0,d1      x:(r1),d7.s ;d2=-Wr*H2i, d1=Wr*H2r-
                                ;Wi*H2i, d7=nBr
fmpy d9,d6,d0 faddsub.s d5,d1    x:(r6)+,d8.s y:.,d9.s
                                ;d0=Wi*H2r, d1=2*Ar',d5=2*Br',d8=nWr,d9=nWi
fmpy d4,d5,d2 fadd.s d2,d0      x:(r0),d1.s d1.s,d6.s
                                ;d2=Br',d0=Wi*H2r-Wr*H2i, d6=2*Ar',d1=nAr
fmpy d4,d6,d2 faddsub.s d0,d3    d2.s,x:(r5) ;d2=Ar', d3=2*Ai',
                                ;d0=2*Bi',save Br'
fmpy.sd4,d3,d3 d2.s,x:(r4)      y:(r1)-,d2.s ;d3=Ar',save Ar',d2=nBi
fmpy d4,d0,d0 faddsub.s d7,d1    d3.s,d6.s y:(r0)+,d3.s
;d0=Bi',d1=nAr+nBr,d7=nBr-nAr=nH2i,d3=nAi
fmpy d9,d7,d0 faddsub.s d3,d2    d1.s,d5.sd0.s,y:(r5)
;d0=nWi*nH2i,d2=nAi+nBi=nH2r,d3=nAi-nBi,save Bi'
fmpy.s d8,d2,d1 d2.s,d6.s,d6.s,y:(r4)+ ;d1=nWr*nH2r,d6=nH2r,save Ai'
_end_split
move y:(r4),d0.s                ;conjugate of last Ai element
fneg.s d0
move d0.s,y:(r4)
endm                             ;split

```

C.11.2 Real FFT for DSP56001/2

Fig. C.38 Real FFT for DSP56001/2

```

;
; This program originally available on the Motorola DSP bulletin board.
; It is provided under a DISCLAIMER OF WARRANTY available from
; Motorola DSP Operation, 6501 Wm. Cannon Drive W., Austin, Tx., 78735.
;
; 1024-Point Real Input Non-In-Place FFT. (test program)
; 34886 clock cycles. Sampling period can be 0.87215ms @ 40 Mhz clock rate
; Use 292 program words,4*512 words for data and 2*(128+256) words for twiddle
; factor
;
; Store EVEN index input data to X memory and ODD index input data to Y.

```

```

; Assume scaling down at input, i.e. all input data are divided by 1024 before FFT.
; The outputs of this real input FFT are twice larger than true values. If the
; original FFT values are desired, scaling up factor should be 512.
;
; 'sincosr' generates twiddle factor for FFT.
; 'bitrevtwd56' sorts the twiddle factor in bit-reverse order.
; The generation and reordering of twiddle factors can be done off-line.
; 'gen56' generates input test signals, delete it if you provide input.
; 'CFFT56' does 512 points FFT.
; 'SPLIT56' extracts 512-point complex values for real input FFT.
; Only DC to Niquest frequency are calculated by this program.
; Input data always starts at location IDATA=0, a 512-complex buffer starts at any
; external memory location, ODATA, is required to hold 256-point output data
; groups.
;
; The output of the FFT replace the inputs started at IDATA.
; X:IDATA contains DC*2 and Y:IDATA contains Niquest*2.
;
;
RFFT56T ident 1,0
        page 132,60
        opt nomd,nomex,loc,nocex,mu

        include 'sincosr'
        include 'bitrevtwd56'
        include 'gen56'
        include 'cfft56'
        include 'split56'
;
;
; Latest revision - Nov. 11 92

reset   equ      0
start   equ      $40
POINTS  equ      512
IDATA   equ      $00
ODATA   equ      $1000
COEF    equ      $800

        sincosr   POINTS,COEF
        gen56     POINTS,IDATA

        opt       mex
        org       p:reset
        jmp       start

        org       p:start
        movep     #0,x:$fffe ;0 wait states
        bitrevtwd56 POINTS,COEF
        CFFT56    IDATA,COEF,POINTS,ODATA
        SPLIT56   IDATA,COEF,POINTS,ODATA

        end

;
; Sine-Cosine Table Generator for rfft56.asm
;
; Last Update 11/11/92
;
sincosr macro points,coef
sincosr ident 1,2
;
;       sincosr- macro to generate sine and cosine coefficient
;       lookup tables for Decimation in Time real FFT
;       twiddle factors. Only points/4 coefficients
;       are generated. For real FFT another points/4
;       coefficients with higher freq. are created.
;
;       points - number of points (2 - 32768, power of 2)
;       coef - base address of sine/cosine table
;              positive cosine value in X memory

```



```

;           positive sine value in Y memory
;
;   8/12/92

pi      equ    3.141592654
freq    equ    2.0*pi/@cvf(points*2)

        org    x:coef-points/2
count   set    0
        dup    points/2
        dc     @cos(@cvf(count)*freq)
count   set    count+1
        endm

        org    y:coef-points/2
count   set    0
        dup    points/2
        dc     -@sin(@cvf(count)*freq)
count   set    count+1
        endm

freq1   equ    2.0*pi/@cvf(points)

        org    x:coef
count   set    0
        dup    points/4
        dc     @cos(@cvf(count)*freq1)
count   set    count+1
        endm

        org    y:coef
count   set    0
        dup    points/4
        dc     @sin(@cvf(count)*freq1)
count   set    count+1
        endm

        endm                                ;end of sincosr macro

bitrevtd56 macro    POINTS,COEF
bitrevtd56 ident    1,2
;
;   bitrevtd - macro to sort sine and cosine coefficient
;               lookup tables in bit reverse order for 56156
;
;   POINTS - number of points (2 - 32768, power of 2)
;   COEF - base address of sine/cosine table
;           negative cosine (Wr) and negative sine (Wi) in X memory
;
; Wei Chen
; July-28, 1992
;
        move    #COEF,r1                    ;twiddle factor start address
        move    #0,m0                       ;bit reverse address
        move    #POINTS/8,n0                ;sincosr use N/4 points data,
                                           ;offset for bit rev. is N/8

        move    #POINTS/4-1,n2
        move    r1,r0                       ;r1 ptr to normal order data
        move    (r1)+                        ;no swap on 1st data
        move    (r0)+n0                     ;r0 ptr to bitrev
        do      n2,_end_bit                 ;does N/4-1 points swap
        move    r1,x0
        move    r0,b
        cmp     x0,b
        jgt     _swap
        move    (r1)+                        ;no swap but update points
        move    (r0)+n0
        jmp     _nothing

_swap
        move    r1,r5
        move    r0,r4

```

```

        move    x:(r1),x0 y:(r5),y0
        move    x:(r0),a y:(r4),b
        move    x0,x:(r0)+n0 y0,y:(r4)
        move    a,x:(r1)+ b,y:(r5)
_nothing
        nop
_end_bit
        endm                                ;end of bitrevtw macro

;*****
;
; Split N/2 Complex FFT(Hn) for N real FFT(Fn)
;
; SPLIT56 macro IDATA,COEF,POINTS,ODATA
; SPLIT56 ident 1,0
;
;
; Fi=0.5(Hi+Hn/2-i*)-0.5j(Hi-Hn/2-i*)W i=0,1,...N-1
;
; Bit reverse input, Normal order output
; This macro amplifies coefficients of FFT by 2.
; If absolute values of spectrum are desired, then scaling up factor is 2^(N-1),
; assuming inputs are scaled by 2^N before complex FFT.
; POINTS is the number of real data /2
; COEF is twiddle factor location other than TF used in complex FFT (see sincosr)
;
;
        move    #POINTS-1,n0                ;number of complex FFT input data -1
        move    #POINTS/2-1,n2              ;loop counter
        move    #ODATA,r0                  ;r0 ptr to Ar=Hi
        move    #COEF-POINTS/2+1,r2        ;twiddle factor start location
        move    r2,r6                      ;r6 -> Wi
        lea     (r0)+n0,r5                 ;r5 ptr to Br & Bi
        move    #IDATA,r3                  ;r3 pointer for A'
        move    r3,r4
        move    n0,r1                      ;r1 ptr for B', r1=B
        move    #POINTS/2,n0
        move    n0,n5
        move    m5,m3                      ;m3 and m1 linear address
        move    m5,m1
        move    #0,m0                      ;bit reverse address
        move    m0,m5                      ;bit reverse address
        move    x:(r0),b                   ;b=Ar0
        move    x:(r5),x1 y:(r0),a        ;a=Ai0,x1=Br
        add     a,b x:(r1)+,x0             ;b=Ar0+Ai0=DC, for ptr reason inc r1
        subl   b,a r3,r4                  ;r4 ptr to temp location
        asl     b y:(r1),a                 ;a=something
        asl     a b,x:(r3)+ y:(r5),b      ;a=Niquist=Ar0-Ai0, save DC,b=Bi
        move    a,y:(r0)+n0               ;save Niq in y:ODATA temp,
        move    y:(r0),y0                  ;y0=Ai

do n2,_end_split
    add y0,b y0,a a,y:(r1)-              ;b=Ai+Bi=H2r,a=Ai, save prev. Bi'
    subl b,a x:(r0),b b,y1               ;a=Ai-Bi=H1i, b=Ar, y1=H2r
    sub x1,b x:(r0)+n0,a a,y:(r4)        ;b=Ar-Br=H2i,a=Ar again,
                                          ;save H1i temp,r0->nA
    subl b,a x:(r2)+,x1 y:(r6)+,y0        ;a=Ar+Br=H1r,x1=Wr,y0=Wi
    mac x1,y1,a b,x0 a,y:(r5)             ;a=H1r+Wr*H2r,x0=H2i,save H1r temp
    macr y0,x0,a y:(r5)-n5,b              ;a=H1r+Wr*H2r-Wi*H2i=Ar', b=H1r
    subl a,b a,x:(r3) y:(r4),a           ;b=H1r-(Wr*H2r-Wi*H2i)=Br',
                                          ;a=H1i,save Ar'
    mac -x1,x0,a b,x:(r1) y:(r5),b        ;a=H1i-Wr*H2i,save Br',b=nBi
    macr y1,y0,a y:(r4),y0               ;a=Wi*H2r-Wr*H2i+H1i=Ai',y0=H1i again
    sub y0,a x:(r5),x1 a,y:(r3)+         ;a=Wi*H2r-Wr*H2i, x1=nBr,save Ai'
    sub y0,a y:(r0),y0                   ;a=Wi*H2r-Wr*H2i-H1i=Bi',y0=nAi
_end_split
    move y0,a a,y:(r1)                   ;save last Bi',conjugate last Ai
    neg a #ODATA,r5
    move #IDATA,r0
    move a,y:(r4)

```

```

move          y:(r5),a
move          a,y:(r0)      ;move Niq. back

endm                      ;split56

```

C.12 REFERENCES

1. A. Papoulis, *The Fourier Integral and its Applications*
2. F. J. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proc. IEEE*, Vol. 66, No. 1, Jan. 1978, pp. 57-84.
3. A. V. Oppenheim and R. W. Schaffer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1975.
4. Kevin L. Kloker, "The Motorola DSP56000 Digital Signal Processor," *IEEE Micro*, December 1986.
5. Motorola Electronic Bulletin Board (Dr. BuB) FFT2C Fast Fourier Transform Routine.
6. EDN's DSP Benchmarks EDN, September 29, 1988.
7. *Fractional and Integer Arithmetic Using the DSP56000 Family of General-Purpose Digital Signal Processors*, Motorola, Inc., Application Report APR3/D.
8. Motorola's DSP Bulletin Board (Dr. BuB) DSP56000 Floating-Point Library.
9. *MC68881/MC68882 Floating-Point Coprocessor User's Manual Publication*, No. MC68881 UM/AD, Motorola, Inc., Austin, TX 1987.
10. *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std. 754-1985 New York, NY: IEEE, 1985.
11. Guy R.L. Sohie and K. Kloker, "A Digital Signal Processor with IEEE Floating-Point Arithmetic," *IEEE Micro*, Vol. 8, No. 6, December 1988.
12. Motorola's DSP56ADC16 Data Sheet.
13. A. V. Oppenheim and C. J. Weinstein, "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform," *Proc. IEEE*, Aug. 1972, pp. 957-976.
14. See ref. [3], pp. 404-418.
15. G. Bergland, "A Fast Fourier Transform Algorithm for Real-Valued Series," *Communication of ACM*, Vol. 11, No. 10, October 1968.

A

- Accumulator Shifter, 55–6
- adaptive FIR filters, 319–41
- Address Generation Unit, 66–70
- Address Register Indirect addressing modes, 76–82
- address register update, 90–1
- addressing modes, 70–82
 - Address Register Indirect addressing modes, 76–82
 - DSP56002 addressing mode summary, 82
 - Register Direct addressing modes, 70–1
 - Special addressing modes, 71–6
- analog design, IIR filters, 236
- Arithmetic instructions, 99–109
- assembling
 - DSP56002 macro, 40–1
 - DSP56002 program, 29–30

B

- band-pass FIR filters
 - design, 168–9
 - implementing, 199–219
- band-pass IIR filters
 - designing and implementing, 262–71
- band-stop FIR filters
 - design, 169–70
 - implementing, 211–19
- band-stop IIR filters, designing and implementing, 271–9
- Bilinear Transformation, IIR filters, 221–2, 224–6
- biquad section, IIR filters, 222, 237–43
- bit manipulation instructions, 113–14
- Bit Manipulation Unit, 61–2
- bit-reversed reorder of input data, FFT, 289
- butterfly computation, FFT, 288–9, 296–305

C

- change command, 11–14
- Control Mode, 128–35
- copy command, 15
- CS4215 multimedia audio codec, 127–35
 - Control Mode, 128–35

D

- data accumulator registers, 48–50
- Data ALU execution unit, 45–62
 - Accumulator Shifter, 55–6
 - Bit Manipulation Unit, 61–2
 - data registers, 46–50
 - Data Shifters/Limiters, 56–61
 - MAC and Logic Unit, 50–5
- data registers, 46–50
- Data Shifters/Limiters, 56–61
- Debug-EVM software commands, 6–15
 - change command, 11–14
 - copy command, 15
 - display command, 10–11
 - help command, 6
 - radix command, 9
- Debug-EVM Software program, 30–1, 33–7
- defining DSP56002 macros, 37–41
- demonstration system
 - FIR filters, adaptive, 334–41
 - IIR filters, 246
- design results, FIR filters, 180–219
- digital design, IIR filters, 237
- Digital Filter Design and Analysis System, 245
- Digital Filter Design and Analysis System
 - software program, FIR filters, 179
- digital signal processing core, 3

digital signal processing systems, 125–56
 CS4215 multimedia audio codec, 127–35
 DSP System, 126–7
 DSP56002 Interrupt Priority Register, 147–51
 DSP56002 processor Port C, 135–47
 exercising DSP System, 155–6
 initialization macro, 151–5
 display command, 10–11
 DSP Assembler program, 25–41
 assembling a canned DSP56002 program, 29–30
 defining and using macros, 37–41
 entering and editing a canned DSP56002 program, 28–9
 installing the ASM56000 Assembler Software program, 25–6
 program development and execution using the Debug-EVM Software program, 30–7
 program source statement format, 26–8
 DSP System, 126–7
 DSP56002, 1–24
 assembling program, 29–30
 Debug-EVM commands, 5–15
 Evaluation Module, 4–5
 general description, 1–3
 installing the software, 5
 using the EVM to program, 15–24
 DSP56002 architecture and addressing modes, 43–83
 Address Generation Unit, 66–70
 addressing modes, 70–82
 architecture review, 43–5
 Data ALU execution unit, 45–62
 DSP56K central processing unit, 44
 expansion area, 44–5
 procedure, 45
 program controller, 62–6
 DSP56002 Demonstration Software, FFT, 289
 DSP56002 Digital Signal Processor. *See* DSP56002
 DSP56002 instruction set, 85–123
 arithmetic, 99–109
 bit manipulation instructions, 113–14
 FIR filters, adaptive, 324–34
 format, 85–6
 implementing FIR filters, 171–7
 logic, 110–13
 loop, 114–17
 move, 97–9
 parallel-moves, 87–8
 Program Control, 118–21
 summary, 121–3
 DSP56002 Interrupt Priority Register, 147–51

DSP56002 macros
 assembling, 40–1
 FIR filters, 177–9
 IIR filters, 245–6
 DSP56002 processor Port C, 135–47
 general-purpose I/O (GPIO), 135–9
 SSI port Control Register "A" (CRA), 140–3
 SSI port Control Register "B" (CRB), 143–4
 SSI port pins, 139
 SSI port Receive Registers, 144–5
 SSI port selection, 139–40
 SSI port Status Register, 145–7
 SSI port Transmit Registers, 145
 DSP56K central processing unit, 44

E

editing DSP56002 program, 28–9
 exercising DSP System, 155–6

F

fast Fourier transform. *See* FFT
 FFT, 283–316
 bit-reversed reorder of input data, 289
 butterfly computation, 288–9, 296–305
 group and coefficient offsets, 305–6
 implementation, 289–317
 macros, 306–17
 review, 284–9
 running DSP56002 Demonstration Software, 289
 storing input sequence $X(n)$ in bit-reversed order, 292–6
 twiddle factors, 290–2
 Filter Transfer function, IIR filters, 222–4
 finite impulse response filters. *See* FIR filters
 FIR filters, 157–219
 demonstration system, 179
 Digital Filter Design and Analysis System software design, 179
 frequency response, 158
 implementation, 158–62, 170–219
 practical FIR filter to ideal FIR filter, 158–61
 software program to design, 165–70
 specifying, 161–2
 window method to design, 162–5
 FIR filters, adaptive, 319–41
 demonstration system, 334–41
 DSP56002 instructions, 324–34
 implementing, 323–34
 LMS algorithm review, 320–3

G

general-purpose I/O (GPIO), 135–9
group and coefficient offsets, FFT, 305–6

H

help command, 6
high-pass FIR filters, 167–8
 implementing, 188–99
high-pass IIR filters, 255–62

I

IIR filters, 221–81
 analog design, 236
 Bilinear Transformation, 224–6
 biquad section, 237–43
 demonstration system, 246
 designing and implementing
 band-pass filter, 262–71
 band-stop filter, 271–9
 high-pass filter, 255–62
 low-pass filter, 247–55
 digital design, 237
 Digital Filter Design and Analysis System,
 245
 DSP56002 macro, 245–6
 Filter Transfer function, 222–4
 implementing, 244
 normalized low-pass filter design, 228–36
 quantization effects, 279–81
 specifications, 227–8
immediate short data move, 87–9
Impulse Invariant Transformation, 221–2
Infinite Impulse Response (IIR) filters. *See*
 IIR filters
initialization macro, 151–5
installing the ASM56000 Assembler Software
 program, 25–6
instruction format, 85–6

L

Limiters, 59–61
LMS algorithm review, FIR filters, 320–3
loading DSP56002 macros, 41
long memory data move, 93–4
loop instructions, 114–17

low-pass FIR filters
 design, 166–7
 implementing, 180–8
low-pass IIR filters, design and implementation,
 247–55

M

MAC and Logic Unit, 50–5
macros
 defining and using, 37–41
 FFT, 306–17
 FIR filters, 177–9
 IIR filters, 245–6
 initialization, 151–5
Momentum Data Systems, 245
Move instructions, 97–99

N

normalized low-pass filter design, IIR filters,
 228–36
 Butterworth design, 232–4
 Chebyshev design, 234–6

O

one-word instruction, 85–6

P

parallel-move types, 87–96
 address register update, 90–1
 immediate short data move, 87–9
 long memory data move, 93–4
 register to register data move, 89–90
 X- or Y-Memory and register data move,
 92–3
 X- or Y-Memory data move, 91–2
 XY-Memory data move, 94–5
Program Control instructions, 118–21
Program Controller, 62–6
program source statement format, 26–8
programming a canned DSP56002 macro,
 38–40

Q

quantization, IIR filters, 279–81

R

radix command, 9
Register Direct addressing modes, 70–1
register to register data move, 89–90
rounding, 53–5

S

saving and loading DSP56002 programs, 17–18
software programs, FIR filters, 165–70
 band-pass filter design, 168–9
 band-stop filter design, 169–70
 high-pass filter design, 167–8
 low-pass filter design, 166–7
Special addressing modes, 71–6
SSI port Control Register "A" (CRA), 140–3
SSI port Control Register "B" (CRB), 143–4
SSI port Receive Registers, 144–5
SSI port selection, 139–40
SSI port Status Register, 145–7

SSI port Transmit Registers, 145
storing input sequence $X(n)$ in bit-reversed order,
 FFT, 292–6

T

twiddle factors, FFT, 290–2
two-word instruction, 86

W

window method to design FIR filters, 162–5

X

X- or Y-Memory and register data move, 92–3
X- or Y-Memory data move, 91–2
XY-Memory data move, 94–5



Digital Signal Processing Applications with

MOTOROLA'S DSP56002 Processor

Mohamed El-Sharkawy, Ph.D.

Motorola's DSP56002 processor and its development tools provide an ideal environment for digital signal processing. This book explains and demonstrates how to use this processor to solve a number of common real-time signal processing problems.

The text has three main sections. Part One covers the operation of the processor and its evaluation module (DSP56002EVM) and assembler. Part Two illustrates the architecture, addressing modes, and instruction set. Part Three takes on the specifics of digital signal processing, including the design and implementation of:

- FIR and adaptive FIR filters
- IIR filters
- Fast Fourier Transforms

This book is intended for use by both students and computer industry professionals. An associated MS-DOS program, *DSP56002 Demonstration Software*, is recommended as an accompaniment to the text. The book includes an order coupon for this software.

Dr. Mohamed El-Sharkawy is a professor of Electrical Engineering at Purdue University. He developed this book in conjunction with Motorola's Digital Signal Processor Division and its University Support Program.



PRENTICE HALL
Upper Saddle River, NJ 07458

For book and bookstore information



<http://www.prenhall.com>

ISBN 0-13-569476-0



9 780135 694763



90000